Монитор

журнал для программистов — практиков

3.93

ISSN 0869-3889

Искусство реконструкции кода

PROGRAMMER

СУБД на Turbo Vision

И надпись написал...

HARDWARE

Микропроцессор Pentium

GAMES

Тайна острова обезьян

AUTODESK

Сначала была идея, затем длинный путь ...
AutoCAD R12 - это 174 шага вперед от R11,
это скорость и мощь "тяжелого" САПРа на 386,
это ворота в высокие технологии.

R12 и R11 полностью совместимы друг с другом;
и русские и английские приложения могут
работать с одной копией R12.
AutoCAD R12 - это SQL мир ...

AutoCAD R12

AutoCAD R12 сделает Вас неуязвимым в шторме инфляции,
знания и мастерство - это самая твердая валюта.
AutoCAD R12 Вам по карману!

Продукция Autodesk распространяется через разветвленную сеть дилеров.
Наши дилеры рядом с Вами! Звоните нам сейчас ...

Autodesk RF

Россия, 107005, Москва,
2-я Бауманская, 9/23

Тел. (095) 261 6363
Факс (095) 267 9927

Есть все основания полагать, что к концу текущего года читатель, жаждущий получать самую свежую информацию о компьютерной и околокомпьютерной жизни, будет иметь реальную возможность выбора среди не менее чем двух десятков весьма разных по содержанию и стилю компьютерных (околокомпьютерных) газет и журналов, выходящих на русском языке.

1) Более или менее регулярно выходящие издания (вышло из печати более одного номера).

Газеты: Софт Маркет, ComputerWorld-Moscow, Человек и компьютер, Компьютерра.

Журналы: PC Magazine/Russian Edition, Мир ПК (PC World/Russia), КомпьютерПресс, Журнал д-ра Добба, Монитор, Сети, Персональные программы, Nantucket News (преобразован в CA News, первого номера которого мы пока не видели), Открытые системы (бюллетень ассоциации групп пользователей ОС Unix), БИТ, Программирование, Информатика и образование, The Pinter FoxPro Letter (переводной).

2) Журналы, у которых вышел первый номер: Index Pro, Компьютеры + программы (оба издаются в Киеве), Монитор-Аспект, Компьютерная графика, Гт (издается фирмой ИнтерМикро), Borland Language Express (перевод с английского одноименного американского журнала).

3) Журналы, выход первого номера которых ожидается в ближайшее время: КомпьюЛог, Read.Me (журнал фирмы Soft-Tronic), Макрос (Macintosh в России).

4) Издания, печатающие объявления и отслеживающие цены на компьютеры и оргтехнику.

Мы также внесли вклад в умножение рядов компьютерной прессы. Как мы и обещали в "Мониторе" № 1 за 1993 год, в апреле вышел в свет первый номер журнала "Монитор-Аспект". В подзаголовок "Монитор-Аспекта" вынесены слова "Журнал для тех, кто работает с компьютерами", и мы надеемся, что он окажется интересен как для читателей "Монитора", так и для гораздо более широкого круга людей, программистами не являющихся, но активно общающихся с компьютерами. Среди материалов первого номера выделим блок статей о последних версиях наиболее популярных у нас текстовых редакторов, подробный обзор возможностей наиболее значительных глобальных сетей передачи данных, услуги которых теперь доступны в России, сравнительный анализ доступных на российском рынке портативных компьютеров с рекомендациями покупателям и обзор современных средств обеспечения безопасности при передаче коммерческой информации по каналам связи.

Появление "Монитор-Аспекта" внесло некоторые изменения в наши издательские планы. В 1993 году выйдут 8 номеров "Монитора" и 3 номера "Монитор-Аспекта". Читатели, подписавшиеся ранее на 10 номеров "Монитора", получат полный комплект и того и другого журнала (всего 11 номеров). Все остальные смогут гарантированно получать "Монитор-Аспект", только подписавшись на него. Подписной купон "Монитор-Аспекта" (и подписной купон "Монитора", если Вы еще не выписали наш журнал) Вы найдете на стр. 43 и 20.

Мы продолжаем отбор и распространение наиболее интересных программных продуктов отечественных разработчиков, список программных продуктов, прошедших наш отбор, Вы можете найти в "Компьютерном ларьке" на стр. 31. В течение 1993 года мы намерены принять участие в нескольких крупных компьютерных выставках; программные продукты из "Ларька" можно будет посмотреть и приобрести не только в редакции, но и на стенде "Монитора" на этих выставках. Ближайшая из них — Международный Компьютерный Форум (1-4 июня 1993 года).

Журнал для программистов-практиков

Редколлегия:
Д.Э.Бройтман
(главный редактор)
О.А.Балдина
В.В.Водолазский
В.С.Головин
Н.Н.Косарев
А.М.Миляков
(ответственный секретарь)
Н.А.Никитина
(отдел рекламы)
А.В.Орехов
(издательский директор)
Е.В.Равич
Л.И.Турок

Графическое бюро:
С.В.Макельский
(главный художник)
Г.Л.Карчикян
(компьютерная верстка)

Отдел распространения:
Р.А.Болдырев
А.А.Винокуров
Е.В.Гун

Представительство в Европе (Германия)
Марина Гетманская
тел. 8-(0631)-7-49-05

В оформлении номера использованы рисунки
А.С.Замулы

Подписано в печать с оригинал-макета 05.05.93
Формат 60x84/8.
Тираж 30 000 экз.
Печать офсетная. Печ. л. 10,0
Цена договорная
Зак.

Издатель:
Научно-техническое общество
Софт·Москва

Для писем: 117419 Москва, а/я 765
Тел. (095) 238-99-33
Факс (095) 238-32-78
Телекс 613322 GRANT SU
BBS (095) 135-14-39 ARSORBIS

© "Монитор", 1993

Письма

"Приют" для хэкеров 4
Как оплачивать интеллект
Тайна клавиши Shift-L

Технологии программирования

А.Штрик. Зарубежные CASE средства создания программного обеспечения в отечественных разработках 6

Рассмотрены зарубежные инструментальные программные средства автоматизированной разработки прикладного программного обеспечения, официально распространяющиеся в России. Приводятся их назначение и область применения, состав и основные возможности, а также аппаратная платформа функционирования.

А. Колесов, О. Павлова. Профессиональная работа в среде Microsoft BASIC Часть 2. Работа с инструментальными библиотеками подпрограмм 14

Возможность разработки сколь-нибудь большой программы во многом определяется наличием готовых библиотек подпрограмм. При работе в системе QuickBASIC эта проблема имеет целый ряд особенностей по сравнению с традиционными компиляторами.

С. Никоненко. Программируем СУБД на Turbo Vision Часть 1 19

Введение в объектно-ориентированную библиотеку DATA VISION для Turbo Pascal версии 6.0+. Приведен исходный текст конвертера файлов dBASE в файл СУБД PARADOX.

Базы данных

А. Пустеленин. CLIPPER и ООП: фантазия или реальность ? 22

В статье предпринимается попытка применить возможности объектно-ориентированного программирования при создании программ на Clipper 5.0.

Мир графики

А. Тихонов. И надпись написал... 32

Рассматривается задача вывода на экран текста, подвергнутого произвольным линейным преобразованиям.

Лицей

Л. Ливеровский. Немного о дисассемблерах. (Опыт работы с программой SOURCER) 40

В статье рассказывается о принципах работы дисассемблера SOURCER и приводится список его команд.

Терроризм

Ю. Косивцов. **Двухкомпонентная антивирусная система** **48**

Описаны принципы построения резидентных антивирусных программ, блокирующих несанкционированный доступ к вычислительным ресурсам.

А. Аброськин. **Резидентный перехватчик функции EXEC** **52**

Приводится текст резидентной программы, перехватывающей одну из важнейших функций DOS, которая может найти очень много полезных применений.

САПР

В. Нефедов, Д. Самсонов **От кульмана к CADKEY...** **56**

CADKEY — графическая система конструкторско-проектных работ для персональных компьютеров. Является альтернативой хорошо известной российскому читателю графической системы AUTOCAD фирмы Autodesk.

Право

М. Корсаков. **Юридическая защита программ в России: практическая реализация законодательства** **64**

В статье рассматриваются основные положения Закона РФ "О правовой охране программ для электронно-вычислительных машин и баз данных" и проблемы практической реализации правовых норм по охране программ.

Игры

Игры с несколькими участниками Часть 2 **68**

Вторая часть обзора, посвященного играм жанра Role Playing Games.

О. Стариков. **Нашел ли ты секрет игры "The Secret of Monkey Island"?** **72**

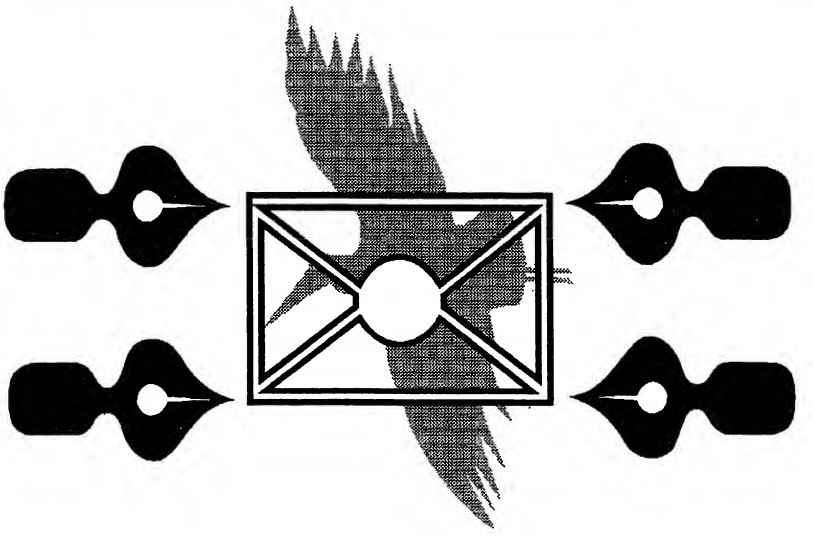
Азартная и увлекательная игра "Тайна острова Обезьян" содержит в себе много тайн и неожиданностей. О том, как разобраться с некоторыми из них, рассказывается в статье.

Аппаратура

Математические сопроцессоры Intel — скорость и точность вычислений **74**

Д. Бройтман. **Микропроцессор Pentium Часть1** **76**

С точки зрения программиста рассматривается новое поколение микропроцессоров фирмы Intel.



"Приют" для хэкеров

"Приют для хэкеров" — под таким интригующим заголовком в газете "Нижегородская ярмарка" (№ 48, 1992) была помещена заметка о создании компьютерного клуба для программистов в Нижнем Новгороде.

"...Нижегородский компьютерный центр пользователей (НКЦП) решил создать компьютерный клуб для программистов, где могли бы встречаться и "умудренные судьбой бойцы с седою головой" и совсем юные хэкеры, освоившие "ТЕТРИС" раньше, чем научились читать.

В клубе можно будет проводить серьезные заседания с докладами и просто полистать "Монитор", "Софтпанораму", (??? — прим.ред.), "Компьютерпресс", "Мир ПК". Все это очень интересно было бы обсудить, обменяться мнениями о современных компьютерных технологиях, о публикациях в расплодившихся софт-журналах..."

Отлично! Наконец-то и у нас. Началось, закрутилось, завертелось, поехало и побежало... Держись, столица! Центр компьютерных технологий автоматически переносится в Нижний Новгород. Жителей города Большие Васюки просят не беспокоиться.

Привет таинственному Б.Н.Н. и прочим сотоварищам!

Ударим автопробегом по бездорожью! Затмим славу "Софтпанорамы" и прочих софт-литераторов... Хэкеры всех стран, объединяйтесь!

Связавшись по телефону с организаторами клуба и получив от них такой заряд бодрости, я, не долго думая, сбрасываю на "винт" адрес вектора НКЦП и устанавливаю свой внутренний таймер на 0Fh января 5:00 p.m.

В назначенный день X маскирую все NMI и, потрепав сервер за ухо, оставляю его в WAIT и делаю малый ход в сторону предстоящего места встречи.

И вот я уже на входе. С трепетом открываю тяжелую стеклянную дверь НКЦП и... о, чудо! Передо мной стоит во всей красе прекрасно сервированный стол. Вино, закуска, шампанское. Молодые люди в пиджаках и белых рубашках с галстуком.

– Хм..., на хэкеров как-то не похоже, скорее на программистов НКЦП. Ничего себе, FindFirst, — подумал я. Вскоре мое присутствие было замечено, и в ответ на мой немой вопрос мне вежливо объяснили, что этот праздник — вовсе не банкет по поводу открытия компьютерного клуба, а просто очередное рабочее заседание сотрудников НКЦП. Желающих общаться ждут по другому адресу. Словом, флаг Вам в руки, товарищ страждущий!

А посему мне было предложено трассироваться пару остановок на любом подходящем троллейбусе. Так сказать, выполнить функцию FindNext.

Привет, ребята! Вы хорошо устроились!

Ну что же, трассировка, так трассировка. Ловлю пробегающий троллейбус и худо-бедно добираюсь до следующего места назначения.

...И вот я опять на входе. Нервишки шалят: если очередной Zero, то больше никаких Find! Еду домой! Однако слышу, слышу... шум, какие-то дебаты..., короче, что-то, отдаленно напоминающее съезд народных депутатов. Храбро открываю заветную дверь и попадаю на общее собрание клуба программистов. Маскируюсь под "чайника" и немедленно подключаюсь к творческому процессу обсуждения ПРАВ и ОБЯЗАННОСТЕЙ членов клуба.

Итак, что же мы ИМЕЕМ и что ДОЛЖНЫ?

ОБЯЗАННОСТИ

Член клуба обязан:

1. Пропагандировать легальное использование ПП.
2. Содействовать работе Учебного Центра по комплектованию учебных групп и проведению занятий.
3. Содействовать проведению семинаров АО НКЦП.
4. Знакомить коллег с новой информацией по ПП.
5. Платить членские взносы в объеме...? в сроки...?

Я не буду рассказывать о самом процессе обсуждения, попробую подвести лишь некоторые итоги, своего рода "сухой остаток". В ходе бурных дебатов были приняты первые четыре пункта без каких-либо изменений. Пресловутый пункт за номером 5 был оставлен для последующего обсуждения во 2-м квартале 1993 года.

Позволю себе прокомментировать этот текст. Лично для себя я вынес следующее. По замыслам организаторов, я как член клуба теперь обязан хватать своего руководителя за шкирку и тащить оного в НКЦП, где ему любезно предоставят возможность купить законную копию того же Norton Commander. А заодно, выполняя п. 2, мне неплохо было бы по пути своего следования собирать зазевавшихся студентов и школьников, тем самым активно содействуя работе Центра по комплектации учебных групп. Не нравится — не ешь. Я, конечно, обеими руками за легальное использование ПО, но не такими же методами.

Да, прав был один мой знакомый, когда на мое предложение посетить это торжище сказал, что не верит никаким НКЦП, так как от них дурно пахнет коммерцией... Вот так. Так что, пардон, ребята. Делайте свою работу сами.

Весьма примечательным мне показался пункт за номером четыре. Как и следовало ожидать, организаторы клуба и тут не дали промаха, дескать, ребята, мы тут все коллеги, братья, поэтому давайте не жмитесь, выкладывайте все свои секреты! Ага, делаю вывод. Значит, я по ночам должен рыть и копать, а потом обязан приносить на блюдечке сухой остаток от HAIR? Неувязочка выходит. Прямо KLUDGE какой-то...

Ну да ладно. Переходим к правам членов клуба.

Как гласила листовка, заботливо подготовленная сотрудниками НКЦП, член клуба имеет ПРАВО:

1. Пользоваться профессиональной литературой.
2. Пользоваться рекламными материалами к новым ПП.
3. Участвовать в подготовке и проведении семинаров АО НКЦП.
4. Пользоваться электронной почтой.
5. Получать помощь при трудоустройстве и поиске заказов.
6. Представлять на обсуждение свои разработки.
7. Участвовать в совершенствовании организационной структуры клуба.
8. Участвовать в апробации и освоении новых ПП (beta-версий) без права их копирования.
9. Получать информацию о проведении конференций и семинаров по ПП и получать приглашения.

Профессиональная литература? О.К. Ловлю на слове. А в ответ — общий привет! Да здравствует продукция НКЦП! Самые лучшие в мире брошюры: "Как войти и выйти из Norton Commander", "Что есть Turbo C, а что не C" и прочие бестселлеры сезона! Забудьте все Guides и References как атрибуты буржуазной пропаганды и идеологии (они нам самим нужны). И как весомый аргумент всем желающим было любезно предложено, "не отходя от кассы", приобрести эти нетленные произведения искусства по сходной цене.

...Особого ажиотажа не наблюдалось.

Пункты 2 и 3 по своей идеологии аналогичны пунктам 1, 2 изложенных ранее обязанностей, поэтому делаем соответствующие выводы. Не стоит обольщаться и по поводу возможного трудоустройства. Наивно полагать, что НКЦП вот так, за просто так найдет Вам подходящую WORK.

Ну а насчет права, так его же у Вас никто не отнимает. У нас в стране у всех все есть. Нет только секса, как сказала одна из легендарных участниц незабвенного телемоста. Кстати, о коммуникациях.

Наиболее догадливые соплеменники вовремя сообразили, что E-mail просто красивое слово, за которым стоит, ну, очень большой ZERO, зато в НКЦП...

Очень интересным мне показался пунктик за номером 8. Это же просто прелесть! Выпускаешь коммерческую beta, обкатываешь ее на "чайниках", можно и на "профи", так даже лучше... И все! Остается исправить найденные bug и убрать слово beta. А при подсчете мегабаксов не забывать подкидывать "чайнику" очередные мегабайты beta. Если честно, идея оригинальная! Но...

Извините меня, господа организаторы. Ковыряться в чужих ошибках... ну, просто нет времени и желания. Пусть подобными апробациями занимается разработчик или, на худой конец, "юные хэкеры" во главе с Павликом Морозовым, а я — wow.

А что же хэкеры?

Хэкеры объявили себя в TSR и не спешили объявиться на свет Божий, предпочитая молча наблюдать за происходящими процессами.

А вокруг страсти, страсти. Основная масса народа ринулась в секцию системного программирования, где предполагалось обсуждать (подумать даже страшно!)... какие-то проблемы СУБД и ООП. Прямо хоть стой, хоть падай! Я так и не понял, при чем тут системное программирование. Wow? Wow? Wow?

Да, как сказал один комик, народу много, а поговорить-то не с кем!

С последней надеждой безуспешно выполняю пару заключительных GROVEL и принимаю решение сделать RET по-английски...

...А на улице ночь. Медленно падал пушистый снег, отражая свет одиноких светофоров, этих немых странников городских дорог...

Праздника не получилось. Успеваю сделать FAR JMP на убегающий вдаль троллейбус. Устало сажусь. Глубоко вздыхаю... Все... IRET...

...А за окном по-прежнему ночь.

...Так же тихо падает снег.

Город спит.
И только хэкер, этот вечный странник
в безбрежном мире информационных технологий,
где-то, наверно, сейчас рядом, успешно ломает
свежеиспеченные CERBERUS и пишет новые SVC...
CATATONIA...

Секретный агент Монитора 007 ■

Здравствуйте!

Я пишу Вам по поводу статьи "Лемминги идут на север" ("Монитор".– 1992.– № 4). В таблице 1 коды уровней для Lemmings не приведены для MAYHEM № 26, 27, 28, 29, 30 — Rendez vous at the Mountain.

За небольшую (в условиях гиперинфляции) сумму 50 тысяч рублей я готов восполнить этот пробел (интеллект должен быть оплачиваем!).

С уважением А.Г. А-ев

Уважаемый господин А-ев!

С сегодняшней почтой я получил Ваше письмо насчет оплаты за раскрытие дополнительных кодов уровней в игре Lemmings. Думаю, что Вы запрашиваете лишнее, а потому делаю Вам со своей стороны контрпредложение и полагаю, что Вы его примете. Мы публикуем Ваше письмо на страницах "Монитора", а Вы платите нам смехотворную (в условиях гиперинфляции) сумму — 15000 (пятнадцать тысяч) рублей. Уверяю Вас, что после того как все читатели "Монитора" (в их числе солидные отечественные фирмы, совместные предприятия, иностранные представительства) узнают о Ваших незаурядных интеллектуальных способностях, Вы получите множество предложений высокооплачиваемой работы.

С совершенным почтением

Д.Э. Бройтман,
главный редактор ■

Уважаемая редакция!

Прочитал в журнале "Монитор" за март 1992 г. статью "Ты записался в убийцы Джафара?" и, естественно, моментально уселся за компьютер, так как в убийцы Джафара я записался давно (не корите за жестокость), а дальше третьего уровня никогда не доходил. Большое Вам, конечно, спасибо, но, сев играть, я обнаружил довольно-таки интересную штуку: случайно спутав комбинацию клавиш Shift-I с комбинацией Shift-L, я совершенно безо всяких усилий попал на следующий уровень.

Так что, если у Вас в редакции установлена первая версия игры "Prince of Persia" (у меня стоит именно она), Вам стоит попробовать поиграть.

Таинственный доброжелатель

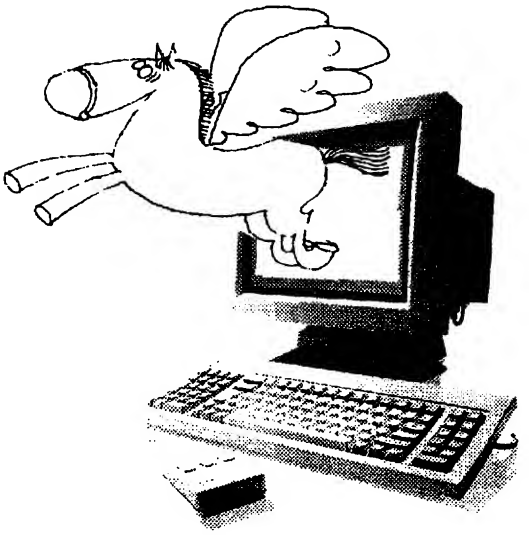
P.S. Вы уж простите, что поступаю так по-свински, но, пожалуйста, напечатайте, можно ли в WINDOWS 3.1 установить русский фонт из WINDOWS 286? И если да, то как это сделать?

Мы попробовали, и все получилось. Поскольку вся играющая братия пребывала в несказанном восторге, то Вам как коллеге простили Ваш проступок. Но все равно вынуждены Вас огорчить. Интересующую Вас информацию получить не удалось. К тому же, как резонно заметили специалисты, пишущие программы для Windows, существует русифицированная версия 3.0, шрифты которой подходят для 3.1. И, кроме того, уже появились программы-русификаторы, например, CyrWin ("Стиплер"), которые поддерживают как традиционные, так и TrueType шрифты для экрана и принтера. Мы, со своей стороны, обещаем опубликовать форматы файлов шрифтов для разных версий Windows, дабы была возможность создать собственную утилиту перекодировки.

В.В. Водолазский ■

Зарубежные CASE средства создания программного обеспечения в отечественных разработках

А. Штрик



Результатом все более тесного сотрудничества с западными партнерами стало появление в России зарубежных CASE средств (CASE — Computer Aided Software Engineering) для автоматизированной разработки программного обеспечения (ПО). Основные концепции, обобщенные технические характеристики, а также перечень официально распространяемых в России зарубежных CASE средств были рассмотрены в [1]. В данной статье приведено более подробное описание конкретных CASE средств из этого перечня, позволяющее заинтересованным специалистам целенаправленно знакомиться с инструментальными средствами и выбирать наиболее полно отвечающие их требованиям. При необходимости еще более детального ознакомления с каждым конкретным CASE средством можно обратиться к приведенной для этой цели библиографии [2]–[14], которая содержит подробное описание инструментальных средств, областей их применения, особенностей использования и другие сведения. Используя перечисленные публикации, нужно учитывать возможность появления новых или улучшенных версий инструментальных средств (приведенные в настоящей статье данные отражают для большинства CASE средств их состояние на момент проведения выставок Софтул-92 и Информатика-92). Ознакомление с самой последней информацией, а также решение всех вопросов поставки, обучения, консультаций и так далее, следует осуществлять через официальных поставщиков соответствующих CASE средств.

Состав инструментальных средств, распространяемых в СНГ

Анализ номенклатуры зарубежных инструментальных CASE средств показывает, что на нашем рынке программные продукты этого класса представлены следующими уровнями интеграции: автономные инструментальные CASE средства; системы программирования третьего поколения (3GL) и языки четвертого поколения (4GL); интегрированные инструментальные CASE средства; интегрированные пакеты. Ниже приведен их перечень по этим уровням интеграции, при этом в скобках указана фирма-разработчик и один из официальных поставщиков данного продукта в нашей стране.

Автономные инструментальные CASE средства

Meta Design (Meta Software Corp., "Стиплер").

Системы программирования 3GL и языки 4GL

Clipper 5.0 (фирма Computer Associates, СП "Магнит");

Clarion (фирма Clarion Software, СП "Инсайт");

Top Speed (фирма Jensen&Partners Int., НПО "Передовые Технологии");

Vaxeln (корпорация DEC, представительство корпорации DEC);

Sapiens (Sapiens Int., НТВЦ Института прикладной математики);

PRO IV (фирма McDonnell Douglas IS, РосНИИ ИТ и АП).

Интегрированные инструментальные CASE средства

Prokit*Workbench (фирма McDonell Douglas IS, РосНИИ ИТ и АП);

CASE-Oracle (фирма Oracle, "ЛогоВАЗ-системы").

Интегрированные пакеты

Base SAS Software (объединение SAS Institute Inc., ЭНПО "Кварк");

Open Access II,III (фирма Software Product Int., СП "Интерсофт");

Object Vision (фирма Borland).

Скорее всего, этот перечень в ближайшее время расширится, поскольку активность зарубежных фирм нарастает, а область инструментальных средств рассматривается ими как достаточно благоприятная для широкого внедрения.

Заметим, что в перечень автономных инструментальных средств не включены такие известные программные продукты как PC Tools и семейство Norton, а в системах программирования отсутствуют широко применяемые компиляторы фирм Borland и Microsoft. Это явилось следствием их массового распространения, в силу чего многие эксперты склонны относить эти продукты к системным программам, а не к инструментальным средствам.

Описание инструментальных CASE средств

Семейство MetaDesign. Инструментальные средства поддерживают использование графического подхода для представления и решения разнообразных прикладных проблем, а также для визуализации процесса мышления. Семейство инструментальных средств включает три компонента.

MetaDesign — средство для проектирования и редактирования моделей сложных систем; представляет собой мощный графический редактор на компьютерах IBM, Sun и Macintosh, позволяющий создавать сложные диаграммы, а также формировать фрагменты текстов на русском и английском языках, связанные по принципу гипертекста. Пользователь проектирует собственную графическую среду, выбирая объекты из стандартного набора MetaDesign или создавая требуемое число объектов с использованием своих представлений. Редактор MetaDesign предназначен как для пользователей-бизнесменов, так и для программистов, инженеров и системных аналитиков проектов. Наиболее эффективные области применения включают проектирование простых и сложных диаграмм; проектирование программных систем методом сверху вниз; представление разнообразной графики; оформление технической документации; планирование и организация работ; документирование текстов программ; описание коммуникационных сетей и так далее.

Design/OA (Design office automation) — инструментальная среда для быстрого создания прикладных графических систем с объектно-ориентированной графикой в области бизнеса и инженерных приложений. С помощью Design/OA программист, работающий на языке C, имеет доступ к Ядру. Ядро содержит подпрограммы и структуры данных, формирующие основу проектируемой прикладной системы. Интерфейс Design/OA содержит более 170 функций, обеспечивающих доступ к Ядру и позволяющих использовать объектно-ориентированную графику. Библиотеки содержат набор функций общецелевого назначения, созданных для того чтобы повысить производительность программирования в прикладных областях, связанных с графическим имитационным моделированием, с анализом переключений и телекоммуникаций, с оценкой производительности управляющих систем и так далее.

Design/IDEF (IDEF — Integrated computer aided manufacturing DEFinition method) — инструментальная система моделирования, автоматизирующая создание моделей данных и SADT (Structured Analysis and Design Technique) диаграмм. Система основана на известной методологии структурного проектирования и анализа сложных систем IDEF/SADT. Design/IDEF автоматизирует стадии проектирования сложных систем различного назначения: формулировка требований и целей проекта, разработка спецификаций, определение компонент и взаимодействий между ними, документирование проекта, проверка полноты и непротиворечивости проекта. При этом пользователям предоставляются следующие возможности: интерактивное обслуживание графики и словаря данных; упрощение описания сложных систем посредством декомпозиции отдельных частей модели; контроль точности, целостности и непротиворечивости модели по мере ее развития; создание нескольких видов отчетов для поддержки и анализа моделей, причем информация из отчетов может быть экспортирована для использования в других программах (издательские системы или электронные таблицы); обеспечение коллективной работы больших групп одновременно работающих над проектом разработчиков. Design/IDEF является наиболее широко используемой в США, Европе и Азии системой моделирования. Среди наиболее эффективных областей применения можно отметить проектирование и анализ широкомасштабных информационных систем; техническое и программное обеспечение вычислительных систем и сетей ЭВМ; управление телекоммуникациями; банковское и биржевое дело; автоматизация производства; анализ технико-организационной работы предприятий; оперативное и перспективное прогнозирование в бизнесе; управление финансовыми ресурсами и материально-техническими запасами.

Платформа функционирования — IBM PC 286/386, PS/2, Compag 386, Macintosh II, Unix Workstations (Sun3/4, Sparc, DEC, HP).

Clipper 5.0 предназначен для создания больших и сложных прикладных систем управления и обработки информации. Является мощным средством проектирования и, поддерживая все возможности предшествующей системы Summer'87, включает мощный препроцессор, новый компилятор, компоновщик и отладчик, дополненную систему переменных, расширенные возможности работы с массивами. Clipper 5.0 имеет открытую архитектуру, позволяющую разработчикам создавать собственные функции и команды, а также встраивать в прикладные программы компоненты, написанные на других языках. Для поддержки профессиональных разработчиков, использующих Clipper 5.0, создана расширенная библиотека инструментальных функций Clipper Tools II/5.0. Она предоставляет программистам, работающим на языке Clipper, области, прежде доступные лишь специалистам, глубоко знающим систему Clipper 5.0 и имеющим большой опыт работы на языках ассемблера или C. Расширенная библиотека включает более 500 оптимизированных функций, которые сгруппированы в определенные тематические направления: справочная информация; примеры программирования; поддержка окон; усовершенствованные драйверы; видеофункции; дисковые утилиты; последовательный интерфейс; сетевые функции; математические функции; непосредственные чтение и запись; синтаксический анализ; преобразования для времени и дат; системные переключатели; разнотипные функции. В перспективе предполагается создание системы разработки объектно-ориентированных программ. Работает на платформе IBM PC и совместимых ЭВМ.

Clarion ориентируется на создание управленческих, организационных и экономических приложений и предназначается для высококвалифицированных программистов, создающих сложные программные комплексы. Основными составляющими являются язык программирования для создания прикладных приложений и набор утилит для поддержки процесса программирования. Система Clarion первоначально создавалась как средство проектирования интерфейсов, однако со временем превратилась в интегрированное средство для создания баз данных и работы с ними. Главной предпосылкой появления Clarion явилось массовое распространение персональных компьютеров, используемых в деловых приложениях, для которых программы, написанные на COBOL, оказались не столь эффективны, как для больших ЭВМ. Clarion — это язык высокого уровня, подобный Pascal, с некоторыми заимствованиями из PL/1, COBOL, Basic. Он отличается компактностью и имеет всего 25 зарезервированных слов.

Набор утилит Clarion, входящих в состав системы, обеспечивает автоматизацию процесса проектирования и программирования базы данных и реализует следующие функции:

- создание файлов Clarion по тексту исходной программы;
- управление файлами Clarion;
- отображение данных на экране в соответствии со структурой базы данных;
- преобразование файлов из формата Clarion в формат dBASE, DIF,ASCII и обратно;
- построение экранов помощи, связанных с полями, меню и другими экранами;
- редактирование текстов Clarion ;
- исполнение и отладку программ в режиме интерпретации с полным набором стандартных возможностей: просмотром и изменением переменных, установкой контрольных точек, трассировкой, повторным выполнением программ;
- создание стандартных объектных модулей, соединение их с библиотечными модулями Clarion и C и создание EXE файлов;
- проектирование приложений, включая создание прототипа прикладной программы;
- создание полностью готовых приложений;
- разработку каркаса (ядра) программы, дополняемого в дальнейшем специфическими процедурами.

Разработанный инструментальный пакет Clarion Professional Developer 3.0 включает интегрированную оболочку, современные средства управления файлами различных форматов, язык программирования Clarion, высокоскоростной оптимизирующий компилятор TopSpeed фирмы Jensen&Partners Int., а также программы, обеспечивающие компоновку с объектными модулями других языков: TopSpeed C, C++, Pascal, Modula-2, Assembler; Microsoft C, Fortran, Macro Assembler; Borland Turbo C, Turbo Assembler. Для доступа к базам данных других форматов (Btrive, Ctree, Fox, Clipper, dBase, Paradox, NetWare SQL) используется технология драйверов баз данных, позволяющая осуществлять одновременный доступ к различным базам данных. Аппаратная платформа — IBM PC и совместимые ЭВМ.

TopSpeed 3.0 — настраиваемая система программирования, включающая высокоэффективные оптимизирующие компиляторы для четырех современных языков, с помощью входящих в нее средств TopSpeed Environtment и TopSpeed TechKit позволяет строить конкретную систему программирования, отвечающую предъявляемым требованиям пользователя. Все компиляторы имеют общий оптимизирующий генератор кода, что позволяет создавать ПО, используя один язык или любую их комбинацию. Кроме того, все языки TopSpeed имеют встроенный планировщик процессов, благодаря чему можно организовать мультипроцессный режим работы в DOS.

Компилятор Modula-2 обеспечивает объектно-ориентированное расширение языка; интерфейс с DOS и BIOS; поддержку виртуальных указателей.

Компилятор Pascal поддерживает полное соответствие стандарту ISO 7185; объектно-ориентированные расширения языка; конвертор исходных текстов (преобразуется более 95% текстов из Turbo Pascal 5.5 и 6.0 в TopSpeed Pascal); включение библиотек Turbo Pascal 5.5 как подмножество библиотек TopSpeed Pascal.

Компилятор C обеспечивает полное соответствие стандарту ISO X3.159; автоматическую генерацию прототипов; совместимость с Turbo C, Microsoft C 5.1

и Quick C; возможность импортировать библиотеки других фирм (Microsoft C, Turbo C).

Компилятор C++ поддерживает полное соответствие стандарту AT&T 2.1; возможность размещения переменных в абсолютных адресах; включение библиотек TopSpeed C; дополнительное включение библиотек Rogue Wave.

Платформа функционирования — среда DOS и OS/2, возможно создание программ для Windows 3 и OS/2 PM.

VAXELN (VAX Ethernet Local area Network) предназначена для разработки прикладных программ реального времени (авиационные тренажеры, управление реакторами, распознавание изображений быстролетящих целей, управление производственными процессами и роботами и так далее). Создание таких прикладных систем традиционными методами без использования эффективных инструментальных средств требовало от разработчиков больших усилий и по проблемно-ориентированному программированию на языках высокого уровня, и по машинно-ориентированному на ассемблере. Система VAXELN позволяет сосредоточиться только на решаемой прикладной задаче, поскольку во время работы прикладной программы все машинно-зависимые операции выполняются ядром и сервисными программами системы VAXELN. Отличительной особенностью системы является поддержка ею доступа к данным в локальной сети Ethernet, что позволяет легко распределять разрабатываемые программы между узлами сети. Система реализует два главных режима работы — инструментальный и исполнительный. Соответственно поставляемый корпорацией DEC комплект содержит как инструментальное, так и исполнительное окружение, которое поддерживает расширенную версию Pascal, а также языки C и Fortran. В состав инструментального окружения входят компиляторы, библиотеки объектных модулей, построитель прикладной программы, отладчик прикладной программы, а также компоненты VAX/VMS: редакторы текста, компоновщик, библиотекарь, файл командных процедур и управляющая программа обслуживания сети. Исполнительная система включает ядро, отладчик, исполнительные библиотеки, программы обслуживания сети и файлов, драйверы периферийных устройств, средства регистрации ошибок. Платформа функционирования — семейства VAX и MicroVAX II под управлением ОС VAX/VMS.

Sapiens — инструментальная система, объединяющая генератор прикладных систем с эффективной и гибкой СУБД. Предназначена для разработки и эксплуатации прикладных систем в таких областях, как промышленное производство; управление запасами и складами; бухгалтерская и финансовая деятельность; страховые компании; системы управления кадрами, транспортными операциями, войсками, центрами ядерных исследований; системы принятия решений. Характерные черты Sapiens: использование базы знаний, осуществляющей поддержку разработки и развития приложений; применение объектно-ориентированного подхода; соответствие стандарту SAA (System Application Architecture) фирмы IBM. В процессе разработки приложений система предоставляет разработчикам следующие возможности: макетирование, динамический анализ и проектирование прикладной системы; автоматическое создание документации; простое и быстрое сопровождение прикладной системы; легкость использования и обучения; дружественный интерфейс с пользователем; механизм защиты данных; полный контроль обработки информации и ее целостность; упрощение функций администратора базы данных.

Жизненный цикл прикладной программы основывается на работе с ее спецификациями, а не с самой программой. Для этого в системе реализован непроцедурный подход, в котором пользователь работает со спецификациями, а синтез машинных программ выполняется автоматически. Сопоставление Sapiens с рядом известных инструментально-программных систем (ORACLE, NATURAL, NOMAD и другими), проведенное экспертами журнала ComputerWorld, показало, что Sapiens занимает лидирующее место по таким важным показателям, как время на разработку прикладных систем; качество документации, автоматически получаемой в процессе работки систем; интегрированность инструментальных средств.

Аппаратно-системная платформа функционирования Sapiens: ЭВМ — IBM 43XX, 30XX,90XX, SIERRA, 370; ОС: OS/MVS, OS/SVS, OS/VS1, VM, DOS/VSE, DOS/VS; совместима с базами данных: DB2, IMS/DB, DL/1, SQL/DS, ADABAS и VSAM; существует интерфейс с прикладными программами на PL1 и COBOL.

PRO-IV — система программирования четвертого поколения — представляет собой интегрированную среду разработки, реализации, исполнения и сопровождения прикладных программ информационных систем на языке PRO-IV. Имеет интерфейсы со стандартными файловыми системами, такими как DL/1, DB2, SQL/DS, IMS, ORACLE, Ingres, Pick, а также с операционными системами VSAM для больших ЭВМ фирмы IBM или RMX для систем DEC и VAX. Прикладные программы на PRO-IV переносимы как по горизонтали, так и по вертикали, что позволяет пользователю выбрать наиболее подходящую среду их исполнения. Например, прикладная программа, разработанная на IBM PC, может быть запущена на исполнение на большой ЭВМ фирмы IBM или на других типах ЭВМ, имеющих систему UNIX, и наоборот. Система PRO-IV обеспечивает создание и модификацию описаний данных и файлов, меню, экранов ввода и вывода, отчетов, прикладных функций; печать данных, файлов и документирование прикладных функций; создание и модификацию описаний правил защиты данных и ограничения доступа к различным утилитам.

PRO-IV автоматически формирует документацию разработки в разрезе файлов, прикладных функций и описаний данных для каждой прикладной программы. Так как система PRO-IV автоматически вносит согласованные изменения в проектные данные, документация всегда соответствует действительности и

готова как для печати, так и для просмотра на экране. В систему PRO-IV включены развитые средства защиты создаваемой прикладной системы. Система PRO-IV работает на технических средствах фирм IBM, DEC, BULL, ICL и других с различными операционными системами; прикладные программы, разработанные на языке PRO-IV, независимы от технических средств, операционной системы и базы данных. Благодаря независимости PRO-IV от среды реализации система PRO-IV требует минимального количества системных ресурсов. Ядру системы PRO-IV необходимо от 250 K до 550 K оперативной памяти.

ProKit*WORKBENCH — интегрированное инструментальное CASE средство для автоматизации проектирования ПО прикладных автоматизированных информационных систем на стадиях стратегического планирования, анализа и разработки программ. Все средства объединяются в базу данных проектирования (репозиторий), накапливающую получаемые в ходе разработки данные. В состав комплекса входят подсистемы:

Анализатор — создание и сопровождение объектно-ориентированной базы данных проекта и схем потоков данных;

Моделирование данных — моделирование структур данных;

Прототипирование — быстрое создание работающей модели системы (экраны, меню, отчеты, запросы, порядок их появления) с целью выявления и уточнения требований к пользовательским интерфейсам;

Проектировщик — разработка и сопровождение структурных схем программ;

Отчетность — подготовка и печать отчетов различного вида (в том числе с использованием издательской системы Ventura Publisher);

Импорт/экспорт — передача данных из репозитория в другие базы данных и обратно.

Система функционирует на IBM PC и совместимых ЭВМ под управлением MS-DOS и может использоваться совместно с системой PRO-IV фирмы McDonell Douglas IS. При этом информация из репозитория ProKit* WORKBENCH экспортируется в систему PRO-IV, что значительно уменьшает трудоемкость последующей разработки. Совместное использование систем ProKit*WORKBENCH и PRO-IV в рамках методологии STRADIS обеспечивает полную поддержку всего жизненного цикла ПО.

Кроме того, ProKit*WORKBENCH включает специальные интерфейсные программы для ее стыковки с известными системами программирования:

TELON фирмы Pansofic Systems, генерирующей программы на языках COBOL, PL-1, COBOL-II для систем IMS/DB, CICS и окружений, работающих в пакетном режиме, образы экранов и отчетов, созданные для прототипа, экспортируются с помощью подсистемы Transport System, входящей в состав системы TELON;

TRANSFORM фирмы Transform Logic Corporation, генерирующей программы на структурном COBOL для систем IMS/DB, CICS и окружений, работающих в пакетном режиме; информация репозитория автоматически перемещается в подсистемы Schema Management System (SMS) и Application Management System (AMS).

Семейство CASE* продуктов ORACLE предназначено для автоматизации проектирования прикладного ПО информационных систем и основано на использовании CASE* метода, позволяющего последовательно отслеживать все этапы жизненного цикла ПО; число таких этапов, перечисленных далее, равно семи.

Выработка стратегии: определяются цели создания системы, приоритеты функций, ограничения, разрабатывается модель системы.

Анализ: строится модель информационных потребностей ("сущность-связь") и модель функциональных требований к системе на основе метода декомпозиции функций; определяются общий план использования системы, области применения и критерии приема системы в эксплуатацию.

Проектирование: подробно прорабатывается архитектура системы, строится концептуальная схема для логического и физического проектирования, после чего выполняется реляционное проектирование базы данных с уточнением ее объемов и показателей производительности обработки. Проектирование программных модулей осуществляется на основе спецификации функций, определенных на этапе анализа. Планируется этап реализации системы и разрабатываются методы тестирования программного продукта.

Реализация: создается реляционная база данных, строятся прикладные системы, проводится тестирование и проверка соответствия программного продукта требованиям пользователя.

Документирование: создается системная документация, материалы для обучения и руководства пользователей.

Внедрение: осуществляется конвертирование данных из прежних систем, если они имелись, и параллельно с эксплуатацией проводится тестирование, анализируются функциональные возможности системы, ее производительность и средства защиты данных от разрушения и несанкционированного доступа.

Эксплуатация: средства автоматизации проектирования используются для поддержки и модификации прикладной системы.

Семейство программных CASE* продуктов включает три компонента.

CASE*Dictionary обеспечивает хранение информации, появляющейся в процессе проектирования информационной системы. Представляет собой словарь данных, в котором хранятся описания информационных моделей, функциональных требований и программных решений. Информационная модель строится на основе отношений "сущность-связь", кроме того, предусмотрено отображение нескольких типов связей. Наряду с ней строится модель потоков данных, которая объединяется с моделью "сущность-связь". В процессе декомпозиции функций формируется их иерархия и строится модель событий, происходящих в системе. На основе модели событий создается сеть модулей, для каждого из которых

формируется спецификация, являющаяся исходной информацией для CASE*Generator.

CASE*Designer поддерживает графический интерфейс при разработке различных моделей предметной области. Имеет доступный пользователю интерфейс, включая систему меню, всплывающие окна, пиктограммы и так далее. Включает также средства для построения четырех типов схем, используемых в проекте: диаграммы "сущность-связь"; диаграммы иерархии функций; диаграммы потоков данных; диаграммы матриц перекрестных ссылок.

CASE*Generator предназначен для автоматической генерации программных модулей; позволяет разрабатывать в виде операторов языка SQL прикладные приложения , обеспечивающие ввод, редактирование и поиск информации в базе данных. Сгенерированные формы отчетов отображаются в спецификациях проектов. В силу полной совместимости исходных текстов ORACLE на всех платформах созданные прикладные приложения могут свободно переноситься между разными аппаратными платформами (например, спроектированная на IBM PC прикладная система может исполняться на большой машине IBM, HP, VAX).

Семейство CASE* продуктов работает более чем на 130 программно-аппаратных платформах, включая большие, мини и персональные компьютеры; для работы необходимо иметь СУБД ORACLE (версии не ниже 5.1).

SAS (Statistical Analysis System) представляет собой интегрированную многофункциональную программную систему для для решения разнообразных задач анализа и обработки данных в различных прикладных областях. Она создана как комплекс отдельных программных продуктов, отличающихся функциональным назначением и связанных единой структурой данных и технологий их обработки. Среди функциональных компонент системы SAS процедуры статистического анализа, реализующие практически все математические методы современной статистики: регрессионный, дискриминантный, дисперсионный, кластерный, факторный анализ и другие; статистические процедуры для анализа временных рядов, применяемые в эконометрике, прогнозировании, финансовом моделировании, надежности; процедуры исследования операций для поддержки принятия решений, оптимизации методами математического программирования, сетевого планирования и так далее; статистические приложения для промышленных задач контроля качества и многое другое.

Основной компонент, обязательный для любых функциональных продуктов системы SAS, — это инструментальное средство Base SAS, представляющее собой интегрированную инструментальную CASE систему. Он включает супервизор, алгоритмический язык четвертого поколения SAS, средства управления базой данных, набор процедур для формирования отчетов, утилиты, полноэкранный текстовый редактор, многооконный экранный монитор, встроенную обучающую систему. Имеются также следующие средства: интерактивный язык матричного программирования, позволяющий математикам, инженерам, статистикам, физикам создавать приложения, используя методы линейной алгебры; пакет процедур для формирования и вывода сложных графических изображений: двухмерных и трехмерных графиков, всевозможных гистограмм и диаграмм различного типа, а также средства картографии; интерактивная система визуализации и анализа данных средствами графики высокого разрешения; язык управления экраном: средства построения меню и окон для разработки дружественных интерфейсов в конкретных приложениях; клавишное меню для работы неподготовленного пользователя (предпринимателя, менеджера) позволяет быстро и просто выбрать данные, проанализировать их и получить результаты в необходимой форме; простые и доступные средства связи между машинами различной архитектуры для распределенной обработки данных; набор интерфейсов для связи системы SAS с популярными системами управления базами данных; пакет макросов и подпрограмм для адаптации или разработки собственных программ на различных языках и включения их в систему SAS в качестве процедур, функций или форматов.

В числе машинно-зависимых компонентов системы SAS представлены специальные средства для разделяемого многопользовательского доступа к данным на большой ЭВМ в системах MVS, VM/CMS; готовые приложения для анализа данных о вычислительном процессе, накапливаемых в системах VMS, MVS, VM/CMS, а также компилятор с языка C для больших ЭВМ, позволяющий создавать объектный код для разных типов ЭВМ и эмулятор терминала большой ЭВМ на персональном компьютере.

Аппаратная платформа работы — IBM (VMS, MVS, CMS), DEC, UNIX, IBM PC (OS/2, DOS); все основные базы данных: DB2, SQL/DS, IDMS/R, DATACOM /DB, ADABAS, ORACLE, INGRES.

OPEN ACCESS II,III — интегрированный пакет, представляющий собой семейство постоянно совершенствующихся программных продуктов для автоматизации учрежденческой деятельности в разных сферах: в промышленности, сельском хозяйстве, на транспорте, для автоматизации научных исследований. Пакет ориентирован как на профессионалов-программистов, так и на специалистов в прикладных областях. Программисты могут использовать его как инструментальное средство для разработки прикладных программ; специалисты — для самостоятельного решения функциональных задач. В состав интегрированного пакета входят следующие основные компоненты.

Система управления базами данных реляционного типа предназначена для создания информационно-поисковых систем различного назначения: справочники и картотеки, учет и контроль и тому подобное. Для поиска и извлечения информации используются языки запросов типа QBE (запрос по образцу) и SQL. Генераторы экранных форм и отчетов обеспечивают удобные механизмы для взаимодействия с базами данных и формирования отчетных документов.

Электронная таблица — основное средство для эффективного решения задач финансового планиро-

вания и прогнозирования, обработки результатов научного эксперимента, инженерных расчетов и бухгалтерских операций. Она включает около 60 различных встроенных функций, позволяющих проводить сложные расчеты.

Универсальная графика позволяет представить цифровую информацию в графическом виде; особенностью является трехмерное отображение информации из базы данных и электронной таблицы. Внутри графика можно использовать до 64 различных диаграмм.

Текстовый процессор предназначен для подготовки и редактирования различных документов; позволяет автоматизировать процесс тиражирования корреспонденции, инициируемый с помощью запроса к базе данных. Имеется система орфографического контроля и корректировки ошибок на русском и английском языках.

"Программер" относится к языкам четвертого поколения и служит для создания прикладных программ различными категориями пользователей.

Рабочий стол (средства электронной оргтехники) дает возможность существенно упорядочить и облегчить выполнение рутинных операций, связанных с планированием рабочего времени. Может быть подключен к телефонной сети или работать в локальной сети, используя встроенные функции электронной почты.

Модуль коммуникации предоставляет возможности доступа к информации, хранящейся на других машинах: персональных, миниЭВМ и больших ЭВМ. Поддерживается широкий спектр модемов и предусмотрен режим автоответчика для эффективного обмена информацией между пользователями системы.

Базовая версия OPEN ACCESS III работает в автономном и распределенном режимах. В распределенном режиме система функционирует в среде локальной сети, управляемой сетевым ПО NetWare и 3 COM+OPEN. Открытая архитектура позволяет расширять функциональные возможности как самой системы, так и приложений путем написания и включения в систему процедур на языке С.

Платформа функционирования — IBM PC, PS/2 под управлением MS-DOS 3.1.

Интегрированный пакет Object Vision — это многофункциональное средство, предназначенное для генерации форм документов, особым образом структурированной информации (бланки, счета, справки и так далее), упрощения ввода и обработки информации и ведения архивов. Пакет основан на принципе наглядного программирования, позволяющем программисту-непрофессионалу создавать свои модули на основе логики работы проектируемой системы. Прикладные программы представляют собой "разумные" бланки документов, способные считывать и обрабатывать данные из различных источников. При разработке конкретного бланка существует возможность составлять его из отдельных элементов: линий, рамок, полей и так далее, выбирая их из меню и перемещая на нужное место. После того как документ скомпонован, можно определить типы отдельных полей. Для этого имеется специальное меню, содержащее поля текста, даты, всплывающего меню, псевдоклавиши.

Принципиальное отличие пакета Object Vision от генераторов экранных форм СУБД состоит в том, что вместо написания формул или операторов вызова процедур для разных полей бланка создается дерево решений, иллюстрирующее логику обработки данных в той или иной макрокоманде. С помощью дерева решений, связанного с конкретным полем документа, можно проводить вычисления, проверять входные данные, запрашивать ввод информации и автоматически заполнять поля. Деревья решений более удобны, чем электронные таблицы, для создания различных коммерческих и финансовых документов. Новый подход упростил процесс программирования этого класса задач.

Object Vision в качестве интерфейсной базы данных сети отличается гибкостью, допуская обмен данными в форматах dBASE, Paradox, Btrieve, ASCII, а также может запрашивать информацию из любой прикладной программы среды Windows и работать в сетевом режиме. Платформа функционирования — IBM PC и совместимые модели в среде Windows 3.0. ■

Литература

1. А.Штрик. CASE: автоматизированное проектирование программного обеспечения//Монитор.– 1992.– № 4.
2. А.П.Сверчков. Организация работ и технология проектирования больших прикладных систем с использованием Clipper 5.0/Семинар "CASE технологии": Сб. материалов, Москва, 1992.
3. Nantucket: серьезный бизнес в России. Интервью с Р.Клафом//КомпьютерУорлд-Москва.– 1992.– № 22.
4. Инструментальные средства для Clipper: Справочная информация//Мир ПК.– 1992.– № 1.
5. А.Смородинский. Clarion — СУБД для профессионалов//КомпьютерПресс.– 1992.– № 1.
6. А.Е.Павлов. Инструментальная система VAXELN//Журнал доктора Добба.– 1991.– № 2.
7. Н.Б.Гуляев, Б.А.Позин. Методология создания информационных систем STRADIS/Семинар "CASE технологии": Сб. материалов, Москва, 1992.
8. Н.Б.Гуляев, С.В.Крюков. Система ProKit*WORKBENCH — средство автоматизации проектирования информационных систем/Семинар "CASE технологии": Сб. материалов, Москва, 1992.
9. Н.Б.Гуляев, С.В.Кокорев. Система PRO-IV — окружение языка программирования четвертого поколения PRO-IV/Семинар "CASE технологии": Сб. материалов, Москва, 1992.
10. М.Михайлов. CASE продукты фирмы Oracle//КомпьютерПресс.– 1991.– № 7.
11. И.Бурлаков. SAS — система получения информации//Мы и компьютер.– 1992.– № 1.
12. Пол Литвин. Object Vision — "разумные" формы документов//Мир ПК.– 1991.– № 4.
13. Л.Струкова. Object Vision: первые впечатления//КомпьютерПресс.– 1991.– № 11.
14. А.А.Александров, А.Е.Милохин. OPEN ACCESS III: интегрированная программная среда для ПЭВМ. — М.: Финансы и статистика, 1992.

Центр Информационных Технологий

Центр информационных технологий "ПАРУС" разрабатывает и внедряет системы автоматизации бухгалтерского учета.

НАЧИСЛЕНИЕ ЗАРАБОТНОЙ ПЛАТЫ:

– расчет заработной платы по основной, совмещаемой и замещаемой должностям;
– определение размеров налогов и удержаний, оформление возврата сумм и перерасчет зарплаты;
– расчет выплат по больничным листам и среднего заработка для начисления отпускных;
– печать расчетно–платежных ведомостей, платежных поручений Сбербанку.

УЧЕТ МАТЕРИАЛЬНЫХ ЦЕННОСТЕЙ:

– ведение номенклатуры основных средств, материалов и МБП, инвентарной картотеки основных средств, остатков на начало месяца;
– регистрация первичных учетных документов и формирование данных по приходу–расходу;
– проводка документов по корреспондирующим счетам;
– формирование итоговых документов (мемориальных ордеров, журналов–ордеров) по выбытию и перемещению материальных ценностей;
– формирование оборотных и инвентаризационных ведомостей по материально–ответственным лицам.

УЧЕТ ФИНАНСОВО–РАСЧЕТНЫХ ОПЕРАЦИЙ:

– учет банковских и кассовых операций;
– учет взаимных расчетов;
– формирование итоговых документов по финансово–расчетным операциям (мемориальных или журналов–ордеров, книги целевых расходов, карточки кассовых расходов, накопительных ведомостей, книги учета депонентов);
– подготовка платежных документов с автоматической отработкой в учете;
– поиск документов и формирование отчетов по заданному условию, справочная отчетность за день с выведением остатков на конец дня.

ГЛАВНАЯ КНИГА И БАЛАНС:

– автоматический перенос итогов бухгалтерских проводок в главную книгу и ее печать;
– формирование и печать баланса предприятия.

БУХГАЛТЕРСКИЙ УЧЕТ ДЛЯ ПРЕДПРИЯТИЙ МАЛОГО БИЗНЕСА:

– подготовка проектов платежных документов;
– учет хозяйственных операций;
– учет взаимных расчетов с поставщиками, дебиторами–кредиторами, подотчетными лицами, покупателями, заказчиками;
– инвентарный учет, амортизационные отчисления;
– групповой учет материальных ценностей (материалы, МБП, продукция, товары);
– формирование ежемесячной отчетности:
ведомость учета наличия и движения материальных ценностей (форма В–2);
ведомости учета расчетов:
с поставщиками (форма В–6);
с подотчетными лицами (форма В–5);
с дебиторами–кредиторами (форма В–7);
с покупателями (форма В–8);
ведомость учета амортизационных отчислений;
– получение справочной информации по наличию и движению денежных средств и материальных ценностей;
– формирование книги учета хозяйственных операций, оборотного баланса и баланса предприятия установленной формы.

Телефон в Москве: 162–68–57.

"Парус" гарантирует немедленный возврат всей суммы, истраченной на приобретение наших систем, если Вы сочтете их использование нецелесообразным

Профессиональная работа в среде Microsoft QuickBASIC

А. Колесов, О. Павлова

Часть 2. Работа с инструментальными библиотеками подпрограмм

Характерной чертой современного программирования является непрерывный рост сложности и объема программ, с одной стороны, и все более жесткие сроки их реализации, с другой. Канули в небытие славные времена, когда реализация программы ограничивалась некоторым расчетным алгоритмом, а срок ее выполнения был 2-3 года. Сегодня всем нужен оконный диалоговый интерфейс, графика, создание баз данных и прочее. И не когда-нибудь, а максимум через 2-3 месяца.

Решение этой проблемы очевидно — программист должен в максимальной степени использовать готовые средства разработки, уделяя внимание в первую очередь тем проблемам, которые за него никто уже не будет решать и которые будут определять уникальную ценность данной конкретной программы.

Формирование своего набора инструментария

Показателем класса программиста является наличие богатого арсенала вспомогательных средств и умение правильно работать с ним. При этом личный набор инструментария программиста формируется и пополняется им индивидуально путем приобретения "фирменных" библиотек, создания собственных, обмена с коллегами. При этом, конечно, желательно не отвлекаться на создание "самострочного" инструментария, а использовать готовый. И все же появление собственных вспомогательных процедур неизбежно в любом случае.

Как это ни странно, для начинающих программистов наибольшую проблему представляет правильное использование именно своих готовых наработок. А принцип формирования инструментальных модулей довольно простой: как только у Вас появилась некоторая программная конструкция, используемая в Ваших программах хотя бы два раза (например, работа с меню), то имеет смысл подумать о таком ее оформлении в виде внешней процедуры, которое бы обеспечило ее многократное применение в будущем.

По этому поводу можно сформулировать несколько советов.

1. Сразу выделяйте модули прикладных и инструментальных процедур. Лучше всего записывайте модули каждой группы в разные директории.

2. Старайтесь объединять в один модуль только группу взаимосвязанных процедур, либо имеющих некоторые общие программные блоки, либо те, которые будут использоваться скорее всего совместно. Имейте в виду, что если хотя бы одна процедура модуля применяется в прикладной программе, то автоматически в нее будет включен этот модуль целиком!

3. При обращении к процедуре используйте обмен данными только через список аргументов — никаких COMMON блоков!

4. Особое внимание уделите качеству процедуры: с ней придется работать довольно долго. Во-первых, не жалейте времени на подробные комментарии и описание обращения к ней. Во-вторых, еще раз посмотрите, насколько оптимальна ее реализация, может быть, ее можно сделать короче, сократить количество переменных и так далее. И наконец, протестируйте ее в разных режимах работы. Все это сократит потери времени в будущем.

5. Хорошо продумайте интерфейс обращения к процедуре, а реализовав его, старайтесь не изменять без крайней нужды — ведь придется корректировать все программы, в которых эта процедура уже использовалась.

Использование Quick библиотек и исходных модулей

Как уже отмечалось ранее, процедуры инструментальных средств могут использоваться при работе в среде QB.EXE в виде исходных модулей или Quick библиотеки. И тот, и другой вариант имеет свои преимущества и недостатки, поэтому необходимо четко представлять, в каких случаях следует их использовать.

Суть проблемы заключается в том, что одновременно с интерпретатором может быть загружена только одна Quick библиотека, размеры которой, естественно, ограничены. Поэтому создание и использование некоторой универсальной библиотеки, включающей "все-все-все", является практически нереальной задачей. Quick библиотеку приходится делать индивидуально для каждой конкретной прикладной программы. И вопрос заключается в том, нужно ли ее вообще создавать, и если нужно, то каким образом.

Наиболее простым и естественным режимом работы является загрузка всех процедур программы в виде исходных модулей:

1) в оперативную память загружаются только те модули, которые нужны для работы конкретной прикладной программы, а не вся библиотека вспомогательных процедур;

2) заголовок инструментальных процедур и имеющийся в них комментарий может заменить их описание, которое необходимо при работе с двоичными библиотеками, — это очень удобно;

3) обеспечивается удобный режим сборки и отладки программных модулей (этот вопрос подробнее рассматривается далее).

Но в этом случае от программиста требуется определенное внимание: нельзя забывать, что исправление инструментальных процедур, как правило, категорически запрещено.

Необходимость применения Quick библиотек определяется следующими положениями:

1) возможностью использования процедур, представленных в виде объектных модулей, в том числе реализованных на других языках;

2) значительной экономией оперативной памяти, вследствие чего появляется возможность создания больших по размерам программ.

Кроме того, за счет сокращения числа исходных модулей при использовании Quick библиотек быстрее выполняются процедуры загрузки программы в среду QB.EXE и создание EXE модулей, а также несколько упрощается операция поиска и выбора нужной процедуры в редакторе.

И тем не менее опыт работы авторов позволяет рекомендовать в качестве основного первый вариант. К созданию Quick библиотек следует приступать только в случае необходимости (например, при появлении проблем с оперативной памятью в среде интерпретатора) и Вашей готовности работы с ними.

Создание Quick библиотеки конкретной программы

Рассмотрим порядок создания Quick библиотеки на примере условной программы TEST.

Сначала загружаем в среду QB всю программу с использованием исходных модулей. Их состав автоматически фиксируется в файле TEST.MAK. Кроме того, нам требуются процедуры, реализованные на ассемблере и представленные в виде двоичной библиотеки ASM (QLB, LIB). Таким образом, запуск интерпретатора будет выполняться с помощью следующей команды:

```
QB.EXE /L ASM TEST.BAS
```

В какой-то момент мы решаем создать двоичную библиотеку для нашей программы. Обычным поводом для этого является появление сообщений "OUT OF SPACE" или "OUT OF STRING SPACE".

Анализируя содержимое MAK файла, выделяем две группы модулей программы — прикладные и инструментальные — и с помощью текстового редактора создаем два новых MAK файла — основной TEST.MAK (старый файл уничтожается) и библиотечный TEST_LIB.MAK. С помощью второго мы создаем соответствующую двоичную библиотеку TEST_LIB (QLB, LIB), загрузив

```
QB.EXE /L ASM TEST_LIB.MAK
```

и используя далее команду Make Library из меню Run.

А разработка самой программы продолжается таким образом:

```
QB.EXE /L TEST_LIB TEST.BAS
```

Следует иметь в виду, что в ходе работы в любой момент времени мы можем перебросить некоторые модули из основного MAK файла в библиотечный (опять накопилось много вспомогательных модулей) и наоборот (появилась необходимость его коррекции или проверки работоспособности), отредактировать библиотеку вне среды QB.EXE и прочее.

Естественно, деление программных модулей на прикладные и инструментальные довольно условно. Главное при этом — надо соблюдать некоторые правила создания Quick библиотеки.

1. Библиотека должна быть замкнутой, то есть не должна иметь ссылок на внешние процедуры, не включенные в эту библиотеку. Возможность четкого выделения библиотеки является хорошей проверкой правильного разбиения процедур программы на модули. Например, если Вы включите некоторую вспомогательную процедуру, на которую имеются внешние ссылки, в главный модуль программы, то выделение какой-либо библиотеки будет просто невозможно.

2. В библиотеку могут включаться все модули, которые в дальнейшем не будут подвергаться коррекции, то есть не только вспомогательные модули, но и прикладные, которые Вы считаете уже вполне отлаженными.

3. В первую очередь имеет смысл выделять в библиотеку те инструментальные модули, с процедурами которых программист не работает в данной прикладной программе. Например, процедуры работы с меню, в свою очередь, обращаются к процедурам оконного интерфейса, которые программист непосредственно не использует. Поэтому, может быть, целесообразнее выделить в библиотеку процедуры оконного интерфейса, а процедуры меню загружать в виде исходных модулей.

Работа с инструментальными программными модулями

Практическая реализация многомодульных программ ставит перед программистом несколько новых вопросов организации своей работы:

1. Как правильно сделать обращение к некоторой инструментальной процедуре в своей прикладной программе?

2. Как найти нужную процедуру в этом море исходных модулей, возможно, записанных в разных подкаталогах?

Мы рассмотрим некоторые варианты решения этих проблем на примере использования комплекса инструментальных средств ALL QB TOOLS, разработанного авторами для языка QuickBASIC версии 4.0 и выше. Сразу отметим, что эти рекомендации имеют достаточно универсальный характер и вполне могут быть использованы при работе с другими комплексами, например, Toolbox, входящим в состав пакета Microsoft BASIC PDS версии 7.1.

Комплекс инструментальных средств ALL QB TOOLS версии 2.87 включает 12 библиотек подпрограмм. Библиотеки размещаются на диске в виде иерархической системы каталогов. В корневом каталоге записан файл PACKING.LST — полное описание всех программных модулей и входящих в их состав процедур, а также ряд вспомогательных программ. Общий объем исходных модулей (не считая описаний, демонстрационных программ, утилит, файлов шрифтов и прочего) составляет 780 Кб.

В состав каждой библиотеки входят исходные модули инструментальных и вспомогательных процедур, демонстрационные программы и примеры реализации различных программных конструкций, перечень подпрограмм библиотеки и рекомендации по их применению. Все модули написаны на языке QuickBASIC, но процедуры обмена с видеопамятью имеют альтернативные варианты программ на ассемблере (скорость работы в случае их применения возрастает в 4-10 раз).

Если Вы интересуетесь вопросами программирования в среде Microsoft BASIC, редакция журнала "Монитор" и Ассоциация пользователей языка Microsoft BASIC просит Вас прислать ответы на приведенную ниже анкету.

1. Конфигурация Вашего компьютера, операционная система
   а) сейчас,
   б) планируете в будущем.
2. С какими языками программирования Вы
   а) умеете работать,
   б) работаете сейчас (в том числе Ваш основной язык),
   в) собираетесь работать в будущем.
3. Версия языка BASIC, с которой Вы
   а) работаете,
   б) собираетесь (хотели бы) работать.
4. Основные черты языка BASIC (Вашей версии) с Вашей точки зрения:
   а) достоинства,
   б) недостатки.
5. Имеете ли Вы необходимую техническую литературу по BASIC?
6. С какими инструментальными библиотеками подпрограмм Вы
   а) работаете,
   б) хотели бы работать.
7. Какие темы по работе на языке BASIC представляют для Вас интерес на страницах журнала?
8. Какие виды поддержки Вашей работы Вам необходимы (предоставление информации, консультации, обучение и т.д.)?
9. Сколько Вам лет, Ваш стаж программирования.
10. Профессия, характер решаемых Вами задач.
11. На кого ориентированы Ваши разработки (для себя, для коллег по работе, реализация в других организациях и пр.).
12. Ваш адрес или другие координаты (для связи с Ассоциацией пользователей MS BASIC).

Анкеты просим высылать по адресу: 107061 Москва, Б.Черкизовская, 6-4-106, Ассоциация пользователей BASIC

Каждая библиотека реализована в виде набора относительно небольших исходных модулей. При этом в программу можно загружать только те модули, процедуры которых действительно используются в ней. Например, раздел библиотеки для работы в текстовом режиме экрана имеет 164 Кб исходного текста, но при работе с наиболее популярными вариантами меню требуется загрузка всего 30-40 Кб процедур (и основных, и вспомогательных).

Помимо более эффективного использования оперативной памяти, такая организация инструментальных библиотек существенно упрощает проблемы коррекции и расширения процедур, в том числе самим пользователем.

Практический пример — обращение к вертикальному меню

Допустим, что в результате изучения демонстрационной программы MENUDEMO.EXE из комплекса ALL QB TOOLS Вы решили воспользоваться процедурой работы с вертикальным меню. Как научиться правильно работать с ней?

Прежде всего с помощью файла PACKING.LST необходимо определить, что библиотека QB-MENU находится в подкаталоге SCREEN0\MENU. Там же можно найти интересующий нас программный модуль — MENU_VER.BAS, в комментариях которого содержится описание состава процедур и обращения к каждой из них. Но самый простой путь научиться работать с процедурой — это посмотреть примеры ее применения и описания в демонстрационной программе MENUDEMO.BAS или специальных демонстрационных примерах в подкаталоге EXAMPLES. Порядок использования данной процедуры в своих программах можно представить следующим образом:

1) загрузите в редактор QB.EXE вместе со своей программой текст демонстрационного примера;

2) найдите в демонстрационном тексте подходящий пример применения процедуры;

3) скопируйте его в свою программу;

4) выгрузите текст демонстрационного примера;

5) исправьте полученный шаблон обращения к меню так, как это Вам надо.

Предположим, что для начала Вы решили использовать самую простую конструкцию меню, и фрагмент обращения к ней выглядит следующим образом (пример).

Следующий шаг работы — загрузить в среду QB.EXE все модули, процедуры которых обеспечивают выполнение написанной Вами программы. Естественно, сначала надо загрузить модуль MENU_VER.BAS, в котором находится процедура MenuVer.

Процедуры библиотеки QB-MENU, в свою очередь, используют вспомогательные процедуры библиотеки оконного интерфейса QB-WNK — WNK_WIND.BAS (вывод окон и текста) и WNK_BAS.BAS (работа с видеопамятью). Информация обо всех дополнительных модулях записана в главной части этого вызывающего модуля, а также в файле PACKING.LST. Но такой способ определения последовательности загрузки программных модулей не очень удобен, так как надо специально смотреть какие-то описания, к тому же, эти модули могли быть загружены уже раньше.

Мы рекомендуем сделать иначе — просто запустить программу на выполнение. Если окажется, что какие-то нужные внешние процедуры не загружены, интерпретатор сам укажет на них. Основная проблема при этом — правильно понять сообщение интерпретатора (иногда оно может сбить с толку, и Вы будете искать причину ошибок не там, где надо), а также найти нужный программный модуль.

Как реагировать на сообщения интерпретатора

Правила работы с внешним процедурами требуют их описания в заголовке вызывающего модуля с помощью операторов:

```
DECLARE SUB      'описание
                 'процедуры-подпрограммы
DECLARE FUNCTION 'описание
                 'процедуры-функции
```

Если процедура находится в том же модуле, что и вызывающая программа, то эти операторы могут не задаваться в явном виде, — они будут сформированы интерпретатором автоматически. Внимательней надо отнестись к описанию процедур, которые размещены в других модулях, рассмотрим именно этот обычный случай.

При работе с процедурами-подпрограммами оператор DECLARE может быть опущен (он также фор-

Пример

```
DECLARE FUNCTION MenuVer%(Row%,Column%,Fore%,Back%,Menu$(),Km%,z$,Kst%)
...
Km% = 3                    ' количество позиций меню
DIM Menu$(1 TO Km%)        ' исходный массив меню
Menu$(1)="Позиция 1"       ' заполнение массива
Menu$(2)="Позиция 2"
Menu$(3)="Позиция 3"
z$ = "Заголовок меню"
Row% = -5                  ' номера строки и столбца верхнего левого
Column% = 10               ' угла окна меню, Row% < 0 — вывод окна с "тенью"
Fore% = 15: Back% = 4      ' цвет текста и фона окна
Kst% = 1                   ' режим автоматического восстановления экрана
                           ' обращение к внешней процедуре-функции:
k% = MenuVer%(Row%,Column%,Fore%,Back%,Menu$(),Km%,z$,Kst%)
                           ' при выходе из процедуры в результате нажатия клавиш:
                                ' Enter: k% = номер позиции;
                                ' Esc: k% = 0.
```

мируется автоматически), но реакция системы на отсутствие нужного модуля будет различной, в зависимости от формы обращения к подпрограмме: с помощью ключевого слова CALL или без него (обращение имеет формат самого оператора языка).

При обращении к процедуре-подпрограмме, если оператор DECLARE SUB отсутствует, а сама процедура не была загружена, выдаются сообщения:

1) с ключевым словом CALL:

"Subprogram not defined" — подпрограмма не определена;

2) без ключевого слова CALL:

"Syntax error" — синтаксическая ошибка.

При работе с процедурой-функцией (в примере с вертикальным меню мы имеем дело именно с ней), находящейся в другом модуле, ее описание в явном виде является обязательным. В случае отсутствия оператора DECLARE FUNCTION обращение к процедуре-функции воспринимается интерпретатором как обращение к некоторому массиву данных, поэтому могут быть выданы сообщения двух типов:

1) "Subscript out of range" — описатель выходит за допустимые пределы (если описание формальных параметров не удовлетворяет обращению к массиву, например, в качестве формального параметра используется символьная переменная, которая никак не может быть принята за индекс массива);

2) "Array not defined" — массив не определен (если описание формальных параметров удовлетворяет обращению к массиву).

Обратите внимание, что в последнем случае система может вообще не обнаружить ошибку, так как BASIC допускает описание массивов по умолчанию. К счастью, это допускается только в главной части модуля, но не в его процедурах, на работу с которыми приходится основной объем текста программы.

Если оператор DECLARE FUNCTION имеется, но вызываемая процедура не была загружена, то выдается такое же сообщение, как при вызове процедуры с использованием ключевого слова CALL, — "Subprogram not defined".

Учитывая тот факт, что писать DECLARE никто не любит, рекомендуем при работе с подпрограммами пользоваться обращением CALL, а формирование DECLARE FUNCTION выполнять следующим образом.

При появлении сообщения, которое говорит об отсутствии оператора DECLARE FUNCTION, нажать клавишу F2 (команда SUBs меню View). Если процедура уже загружена в память, то мы сразу попадаем в окно искомой процедуры, после чего надо скопировать строку ее наименования в заголовок вызывающего модуля и дописать ключевое слово DECLARE. Если процедуры нет в оперативной памяти, то выполнить загрузку нужного модуля и повторить запуск программы.

Поиск нужного модуля

Вы определили, что программа требует загрузки некоторой процедуры. Но как определить, в каком именно исходном модуле она находится? И где хранится сам модуль? Для его решения мы включили в состав комплекса QB TOOLS специальную утилиту QBFIND. EXE, которая по имени процедуры или модуля выдает краткую справку о составе модуля, входящих в него процедурах и его расположении на диске. Обращение к программе QBFIND.EXE можно сделать прямо из среды QB.EXE либо используя окно "Immediate" и оператор SHELL, либо выйдя на время в MS-DOS по команде DOS меню File.

Будьте внимательны при использовании Quick библиотек

Важным достоинством работы с исходными модулями процедур является то, что их работа автоматически контролируется средствами отладчика интерпретатора. В частности, при запуске программы производится проверка соответствия описания формальных параметров в вызывающем модуле и вызываемой процедуре. В случае несоответствия будет выдана диагностика:

"Parameter type mismatch" — несоответствие типа параметра.

При несоответствии типов данных при обращении к процедуре функции появится сообщение:

"Duplicate definition" — повторное определение функции.

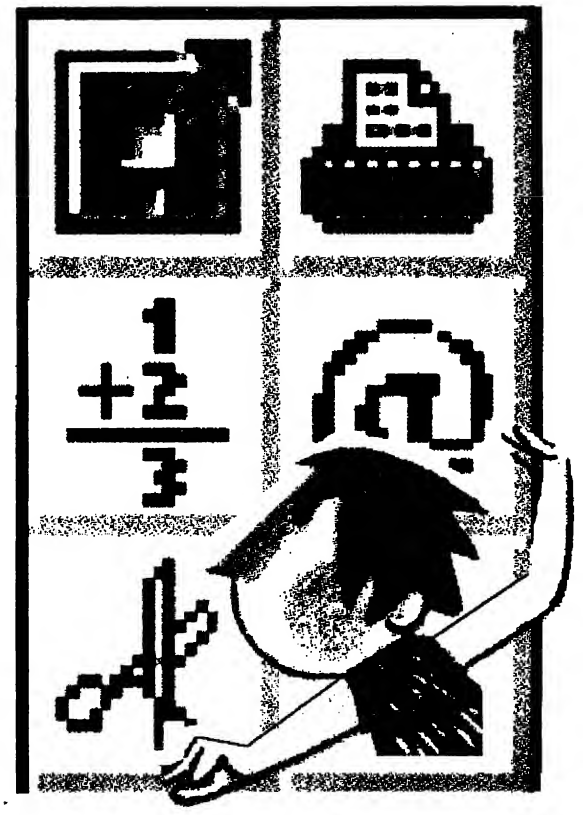
Такие ошибки часто возникают, например, из-за неправильного обозначения типов переменных и процедур-функций (некоторая путаница может появиться при определении типов переменных в явном виде, например, с помощью оператора DEFINT).

При использовании Quick библиотек такого контроля, естественно, не производится, поэтому Ваша ошибка при обращении к процедуре может привести к непредсказуемой реакции системы — чаще всего к ее зависанию. То же самое может произойти при ошибках, связанных с использованием недопустимых значений формальных параметров в вызывающей программе.

В подобных случаях следует внимательно проверить все обращения к процедурам Quick библиотеки, а в случае необходимости — перейти к загрузке процедур в виде исходных модулей для контроля их работы в среде интерпретатора. Разумеется, в этом случае приходится производить соответствующую корректировку Quick библиотеки.

* * *

Возможно, описание процесса сборки необходимых программных модулей покажется читателям несколько сложным и запутанным, но это не совсем так. Любая технология покажется сложной, когда ее осваиваешь в первый раз. На самом деле все происходит гораздо проще: количество практически используемых инструментальных процедур не очень велико и пары часов, посвященных работе с конкретной библиотекой, вполне достаточно, чтобы научиться быстро ориентироваться в ней. ■



Программируем СУБД на Turbo Vision

С.Никоненко

Часть 1

В смятении Жоан Смельчак топтал и рвал траву, ведь должен же он был отыскать... впрочем, что именно, он и сам толком не знал.

Жозе Гомес Феррейра. Чудесные приключения Жоана Смельчака

Небольшое введение

Объектно-ориентированная библиотека Turbo Vision для DOS компиляторов фирмы Borland с самого первого момента своего появления будоражит умы отечественных программистов. Однако ввиду специфики нашего рынка программных продуктов львиную долю на нем занимают так называемые базы данных (БД) — статистические программы, бухгалтерские системы, офисные и банковские приложения и так далее. Такой расклад означает, что доминирующее положение среди прикладного программного инструментария занимают именно СУБД — Clipper, Paradox, Clarion, FoxPro, FoxBase и dBase. Известно множество попыток создания собственного инструмента проектирования БД на языках высокого уровня. В одном случае это заимствование всем хорошо известного dbf формата, в другом — использование Paradox Engine или dbVista.

Познакомившись впервые с объектно-ориентированным программированием (ООП) для DOS и Turbo Vision, многие пытаются приспособить мощь и гибкость объектов этой библиотеки к потребностям проектирования БД. Результаты, как правило, оставляют желать лучшего — нет смысла "изобретать велосипед", создавая собственный формат хранения информации, либо нет возможностей для написания полноценного интерфейса с форматами коммерческих СУБД. Поэтому, как правило, найденные решения не выходят за круг их разработчиков. Автору известны по крайней мере две "импортных" библиотеки для работы с dbf форматом на Turbo Pascal: TPDB и GSDBASE, которые, мягко говоря, во многом некорректны.

По этим причинам статья преследует несколько целей:

- изучить возможности написания БД с помощью объектно-ориентированного подхода Turbo Vision;
- продемонстрировать способности стремительно набирающего популярность объектно-ориентированного инструмента проектирования реляционных БД — Data Vision.

Что такое Data Vision

Название Data Vision происходит от двух англоязычных идиом — Data Base (база данных) и Turbo Vision. Если говорить по существу, Data Vision является объектно-ориентированной библиотекой, позволяющей быстро создавать прикладные СУБД реляционного типа, использующие не зависящий от физического формата данных интерфейс. Фактически речь идет о виртуальных средствах работы с базами данных, которым, в принципе, безразлично, с какими физическими форматами данных работать: dbf файлами (Clipper, FoxBase, dBase), db файлами

(Paradox), dat файлами (Clarion) и так далее или со всеми одновременно.

Еще одна конкурирующая СУБД?

В каком-то смысле, да, но только как инструментальное средство. Data Vision адресована тем программистам БД, которых не удовлетворяют технические (быстродействие, размер загрузочного модуля или отсутствие оного, соответствие стандартам CUA и так далее) и потребительские (эстетичность приложений, их мобильность, гибкость) характеристики прочих СУБД; тем, кому не по душе избыточность и громоздкость приложений на Clipper и Clarion; тем, кто хочет испытать себя в бурных водах технологии ООП; тем, кто любит Turbo Pascal; тем, кто пишет сложные и большие СУБД; наконец, просто любопытным программистам.

Идея о создании Data Vision родилась из мелочей. Необходимость постоянного чередования инструмента программирования при работе над БД и задачами вычислительного характера; постоянные информационные конфликты при работе с неоднородным по формату данных ПО; стремление работать на "настоящем" языке программирования — все эти факторы сыграли роль.

Но главная причина — отсутствие подлинного унифицированного объектно-ориентированного интерфейса с известными базами данных.

Как ООП является новой технологией для старых языков программирования, так и Data Vision нельзя назвать новой СУБД, но новой технологией программирования для СУБД.

И технологией передовой.

Общие отличия

Принципиальное отличие Data Vision от прочих технологий проектирования БД состоит в том, что она реализована на универсальном языке высокого уровня и для него со всем комплексом вытекающих из этого факта последствий: максимальная открытость, гибкость, высокая скорость кода, мобильность, развитая система представления данных.

Объектная ориентация, естественно, включает средства инкапсуляции, наследования и полиморфизма.

Кроме этого, существует возможность использования большого количества уже готовых объектов Turbo Vision: видимых элементов (окон TWindow, диалоговых окон TDialog, полос прокрутки TScrollBar, кнопок TButton, визуальных списков TListViewer, строк ввода TInputLine, кластеров TCluster, разного рода редакторов и селекторов), полиморфных коллекций TCollection, TSortedCollection и TStringCollection, полиморфных потоков TStream, TDosstream и TBufStream, файла ресурсов TResourceFile. Data Vision значительно расширяет дерево объектов Turbo Vision. В дальнейшем мы подробнее остановимся на дереве объектов библиотеки.

Сюда же можно добавить механизм ввода и обработки событий видимыми элементами, как у приложений Windows, стандартизованный пользовательский интерфейс (в соответствии с предложенным стандартом Common User Access фирмы IBM) и глобальную поддержку "мыши".

Все перечисленное является качеством инструмента, на котором написана сама Data Vision.

Data Vision включает следующие принципиальные моменты, рассматриваемые далее.

1. Унифицированный интерфейс с базами данных.
2. Виртуальную базу данных.
3. Элементы электронных таблиц.
4. Пользовательскую функцию.
5. Визуализацию данных.
6. Единую информационную среду.
7. Механизм фреймов.

Первый полезный пример на Data Vision

Прежде чем подробно останавливаться на особенностях реализации Data Vision, приведем небольшой пример ее использования вне контекста видимых элементов Turbo Vision и Data Vision. В примере создается неиндексированная база данных в формате dBase из файла в Paradox формате.

В следующих примерах будет описано использование видимых элементов Data Vision. ■

Монитор — 1993

Подписной купон

1	2	3	4	5	6	7	8

МОНИТОР

Адрес _____

Ф.И.О. или название организации _____

Стоимость подписки на 1 номер журнала "Монитор" — 250 рублей. В 1993 году будет выпущено 8 номеров (годовая подписка — 2000 руб.). Отметьте крестиками те номера журнала, которые Вы хотите получить. Перечислите сумму оплаты на счет НТО "Софт-Москва": р/с 467002 в КБ "Банк развития XXI век", к/с 161816 в РКЦ ГУ ЦБ РФ по г. Москве, МФО 201791.
Вышлите подписной купон и квитанцию об оплате по адресу: 117419 Москва, а/я 765.

Пример 1

```
{ ---------------------------------------------
  Простой пример использования Data Vision
      Преобразователь Paradox —> dBase
      ***
      CopyRight (C) 1993, Никоненко С.Б.
  Лаборатория программных средств SHIFT

            SIMPLE.PAS
--------------------------------------------- }
{$V-}
Uses
  Dos,
  DVDefine,DVDefDB,     { Модули Data Vision v1.0          }
  DVdBase,              { Модуль DVision DBF Interface v1.0 }
  DVPX;                 { Модуль DVision PX Interface v1.0  }

Const
  ParaID   = 1;         { Идентификатор БД в формате Paradox }
  dBaseID  = 2;         { Идентификатор БД в формате dBase   }
Var
  ParaDB, dBaseDB  : pDefDB;     { Указатели на объекты БД  }
  D                : PathStr;
  N                : NameStr;
  E                : ExtStr;

(* Процедура формирует список записей заголовка БД        *)
(* из заголовка базы, указатель на которую DBItem         *)
(* передается в качестве параметра                        *)

Function CreateFromDB (DBItem : pDefDB) : pDBHeader;
Var
  i,Len,Dec : Byte;
  ptr       : ^pDBHeader;
  ptr0      : pDBHeader;
  Ftype     : Char;
begin
  ptr: = @ptr0;
      { Метод возвращает количество полей БД, связанной с объектом }
  for i: = 1 to DBItem^.TETotalFld do
  begin
      { Метод возвращает формат i-го поля :
      FType     - тип поля;
      Len       - длина поля;
      Dec       - количество знаков после запятой }
      DBItem^.TEGetFldForm (i,Ftype,Len,Dec);
      { Функция формирует связанный список имен и форматов полей }
  ptr^ : = NewDBField(DBItem^.TEFldName(i),FType,Len,Dec,nil);
  ptr : = @ptr^^.next
  end;
  CreateFromDB : = ptr0
end;

(* Процедура заполнения одной БД из другой *)
Procedure CopyBase;
Var
  i : Byte;
begin
  With ParaDB^ do
      { Метод TEEOF возвращает True, если пытаемся выйти  }
      {          за нижний предел БД                      }
  While not TEEOF do
  begin
      { Метод добавляет пустые записи к БД                }
      dBaseDB^.TEAddRec(1);
      { Методы чтения и записи данных по формату строки.
      Примечательно, что данные записываются в формате поля,
      несмотря на формат входной информации               }
      for i: = 1 to TETotalFld do
          dBaseDB^.TEPutStr(i,TEGetStr(i));
      { Метод сохраняет запись БД                         }
      dBaseDB^.TESaveRec;
      { Метод перемещается к следущей записи БД }
      TEGoNext
  end
end;


(* Процедура останова программы                           *)
Procedure ErrorHalt (S : String);
begin
  Writeln(S);
  Halt(0)
end;

(* В командной строке задается имя файла                  *)
begin
  if ParamCount < 1 then
  ErrorHalt('Формат командной строки: SIMPLE <Имя Paradox файла>');
  EnginePath : = '';          { оверлей PXENGINE.OVL в текущем каталоге }
  Writeln('Программа SIMPLE преобразования Paradox —> dBase');
  Writeln('CopyRight (C) 1993, SHIFT Lab.');
  InitParadox(@ParaDefault);  { инициализация PARADOX ENGINE  }
                              { с параметрами по умолчанию    }
  if not PXInstalled then Halt(0); { останов, если инициализация не прошла}

  { Инициалйзация объекта физической БД в формате Paradox   }
  { 1 параметр ParaID - уникальный идентификатор БД         }
  { 2 параметр из командной строки - имя файла Paradox      }
  { 3 параметр 0 - признак индекса                          }
  { 4 параметр False - отказ от сохранения каждого изменения БД }
  { 5 параметр 0 - количество полей, которые участвуют      }
  {              в первичном индексе                        }

  ParaDB: = New(pDefPX,Init(ParaID,ParamStr(1),0,False,0));
  if (ParaDB = nil) or (DBInitError > ieExist) then
      ErrorHalt('Ошибка открытия ' + ParamStr(1));
  FSplit(ParamStr(1),D,N,E);
      { Функция возвращает True, если файл БД создан успешно }
  if not DBFCreate(D + N + '.DBF',CreateFromDB(ParaDB)) then
      ErrorHalt('Не могу создать ' + D + N + '.DBF');

  { Инициалйзация объекта физической БД в формате dBase     }
  { 1 параметр dBaseID - уникальный идентификатор БД        }
  { 2 параметр - имя файла dBase                            }
  { 3 параметр '' - имя индекса dBase                       }
  { 4 параметр 0 - номер ключевого поля                     }
  { 5 параметр NoUnique - признак дублирования ключей       }
  { 6 параметр 0 - если = True у индексированной БД, то идет }
  {          реиндексация при инициализации объекта         }

  dBaseDB: = New(pDefDBF,Init(dBaseID,D + N + '.DBF','',0,NoUnique,False));
  if (dBaseDB = nil) or (DBInitError > ieExist) then
      ErrorHalt('Ошибка открытия ' + D + N + '.DBF');
  CopyBase;
  Writeln('Преобразовано ',ParaDB^.TETotalRec,' записей');
  DoneDBList;         { Освобождение памяти всех БД         }
  ExitParadox         { Освобождение памяти PARADOX ENGINE  }
end.
```

CLIPPER и ООП: фантазия или реальность?

А. Пустеленин

Вступление

Вряд ли можно сейчас найти программиста, который бы не стремился хоть как-то использовать преимущества, которые предоставляет объектно-ориентированное программирование (ООП) и уж тем более — программиста, не слышавшего термина "объектно-ориентированное программирование". Об объектно-ориентированном программировании говорят и пишут решительно все. Но увы, не каждый программист может похвастать своим опытом работы в этой области. И тем более, если в процессе Вашей повседневной деятельности Вы не сталкиваетесь с такими мощными языковыми средствами, базирующимися на принципах ООП, как Smalltalk, C++ и другие. Что делать, если основным Вашим языком программирования является CLIPPER? Что может предложить именно сейчас фирма Computer Associates пользователям CLIPPER v5.0 и v5.01?

CLIPPER как объектно-ориентированный язык

В новой версии CLIPPER 5.0 фирма Computer Associates (CA) предложила пользователям ряд средств, позволяющих почувствовать вкус от использования преимуществ, которые предлагает концепция ООП. Правда, эти средства пока ограничивают действия программиста в рамках небольшого набора уже "готовых к употреблению" предопределенных классов, но самое главное есть аппарат для доступа к объектам как таковым. Если копнуть глубже и перелистать подшивки периодических журналов, издаваемых сейчас, то можно наткнуться на статью А.М. Рытова. Она опубликована во 2-м номере журнала "Мир ПК" за 1992 год. В ней мы обнаружим ряд недокументированных функций, которые раскрывают нам секреты создания объектов, доступа к ним, даны наброски возможных команд для описания классов и даже сделана попытка создания полноценной иерархической структуры классов (основанной, подчеркну, на наборе предопределенных классов, только более низкого уровня).

Используя эту статью и проведя ряд экспериментов со стандартными объектами CLIPPER, можно сделать ряд выводов:

- объект — массив данных, с которым ассоциирован словарь методов;
- исполняющая система CLIPPER имеет список описателей классов, в котором хранятся описатели всех динамически (!) создаваемых в процессе выполнения программы объектов;
- существует аппарат доступа к внутренним переменным и экспортируемым методам объекта, созданного на основе описания класса из списка описателей классов.

Используя эту поддержку в рамках системы CLIPPER 5.0, функционирует набор стандартных классов: Get, Error, TBrowse, TBColumn, на основе которых программист может создать свой объект и использовать внутренние переменные и экспортируемые методы для решения своих задач. Но расширять, дополнять или переопределять компоненты классов невозможно!

До этого момента я не сказал ничего нового для многих пользователей, которые используют систему CLIPPER версии 5.0. Но давайте разберемся, что же мы хотим получить от языка программирования, чтобы назвать его полноценным объектно-ориентированным языком, не ограничиваясь при этом только механизмом создания объектов из набора предопределенных классов.

Познакомимся поближе

Попробуем определить свойства объектно-ориентированного языка программирования (ООЯП) и создать аппарат, который превратит

наш CLIPPER в полноценный объектно-ориентированный язык программирования.

Напомню три основных свойства, при наличии которых можно говорить, что некоторый язык предназначен для объектно-ориентированного программирования: инкапсуляция, полиморфизм и наследование.

Инкапсуляция означает скрытие реализации объекта (данных) от использующих его модулей. Некоторые компоненты вновь созданного объекта не могут быть изменены модулями, вызванными вне объекта. Кроме того, развитый механизм инкапсуляции позволяет скрывать наследуемые данные и методы внутри "дерева наследования" (см. "Наследование").

Полиморфизм означает способность объектов выбирать метод, исходя из типа данных, принятых в сообщении.

Наследование описывает механизмы действия языка при построении новых классов из уже имеющихся (построение дерева наследования). Новые классы должны содержать только коды и данные для изменяющихся методов; коды и данные, которые переходят неизменными от порождающих (родительских) классов, остаются в этих классах.

Существует еще ряд свойств, которые либо следуют из трех основных, либо просто входят в подмножество этих свойств. Все эти свойства мы разберем в процессе рассмотрения трех основных свойств.

Теперь, имея в руках минимальный набор свойств, попробуем создать полноценный объектно-ориентированный язык программирования, добавив несколько команд к уже имеющемуся аппарату доступа к объектам.

Объявления классов и создание объектов

Предварительно рассмотрим упрощенные команды для создания просто пользовательских классов, по аналогии со стандартными классами системы CLIPPER, не вводя те свойства, о которых мы упоминали в предыдущей главе.

Прежде всего, хотелось бы отметить, что для полноценного ООЯП необходимо иметь доступ к описателю тела класса с входящими в него компонентами и функциями-компонентами. Поэтому возникает потребность в создании команд, которые могли бы описать внутреннюю структуру создаваемого класса. Так мы подходим к командам объявления класса.

Классы и их объявления

Контекст объявления класса описывает внутреннюю структуру объектов, создаваемых на основе этого класса. Следовательно, класс — это новый тип данных — объект. Каждый класс представляет собой уникальное множество компонентов и функций-компонентов (методов), а также преобразований, используемых для создания и манипулирования такими объектами. Могут быть объявлены производные классы, наследующие компоненты одного или более базовых классов.

Представим первоначально упрощенный синтаксис объявления класса следующим:

```
CLASS <Имя_класса>
   ...
   // Описание компонентов класса
   ...
END CLASS
```

Контекст имени класса переопределяется препроцессором системы CLIPPER в глобальную функцию, доступную из любой точки программы. Таким образом, объявление класса может находиться в любом месте текста программы и не должно встречаться в контексте описания других функций. Контекст имени класса начинается с точки определения CLASS и заканчивается командой конца описания класса — END CLASS. Внутри контекста следуют списки компонентов класса и функций-компонентов класса.

Оставим на время описатели классов и рассмотрим команды для создания компонентов и функций-компонентов, экспортируемых в описываемый класс.

Список компонентов

Список компонентов представляет собой последовательность объявлений данных любого типа, начинающихся с ключевого слова INSTANCE. Список компонентов характеризует внутреннюю структуру создаваемого объекта.

Представим первоначально упрощенный синтаксис объявления компонента следующим:

```
INSTANCE <Имя_компонента> [ := <Значение> ]
```

Контекст компонента, устанавливаемого в класс, переопределяется препроцессором системы CLIPPER в вызов функции, добавляющей <Имя_компонента> и первоначальное <Значение> в список описателя класса. При этом <Имени_компонента> ставится в соответствие указанное <Значение>, по умолчанию значению компонента ставится в соответствие NIL.

Функции-компоненты

Список функций-компонентов представляет собой последовательность объявлений методов, экспортируемых в класс и начинающихся с ключевого слова METHOD. Функции-компоненты предназначены для манипулирования внутренними данными объекта, созданного из этого класса. Методу доступны данные объекта без явной передачи их в качестве параметров.

Представим первоначально упрощенный синтаксис объявления функции-компонента следующим:

```
METHOD <Имя_функции>( [<Список_параметров>] )
```

Контекст функции-компонента, экспортируемой в класс, переопределяется препроцессором системы CLIPPER в вызов функции, добавляющей <Имя_функции>-компонента в список описателя класса.

Отметим, что круглые скобки в контексте объявления метода необходимы для определения списка аргументов, передаваемых методу.

Однако при таком описании всегда есть опасность наткнуться на имя функции, которое уже использовано. В этом случае опишите контекст реализующей

функции в одном программном блоке с описателем класса и укажите область видимости реализующей функции только в этом же программном блоке (описание STATIC FUNC).

Либо используйте следующий синтаксис объявления функции-компонента:

```
METHOD <ПсевдоИмя>( [<Список_параметров>] ) TO ;
       <Имя_функции>
Здесь <ПсевдоИмя>  — имя экспортируемого метода;
     <Имя_функции> — имя функции, реализующей
                     действия метода.
```

При обращении к <ПсевдоИмени> экспортируемого метода будет вызываться функция, реализующая действие метода, имеющая имя <Имя_функции> и принимающая в качестве аргументов <Список_параметров>.

Для доступа функций-компонентов, реализующих методы класса, к скрытым в объекте данным необходимо использовать одну из псевдопеременных, определяемых как

```
#xtranslate :: => _qSelf()
#xtranslate Self => _qSelf()
```

Функция _qSelf() будет автоматически возвращать ссылку на тот объект, которому послано сообщение, и функции-компоненты, реализующие методы, получат доступ к его данным.

Объекты классов

Подводя черту, можно сказать, что класс — это структура, описывающая новый тип данных, на основе которого создаются конкретные экземпляры этого типа — объекты, имеющие определенную внутреннюю структуру и обладающие набором определенных методов. То есть класс является описателем типа данных "объект". Объекты могут существовать, изменяться, выполнять определенные действия.

Теперь мы имеем представление как о понятии класса, так и об объектах класса. Но каким образом создаются объекты?

Конструкторы

Сразу дадим определение конструктора. Конструктор — процедура (функция), которая создает экземпляр объекта указанного класса. Во многих ООЯП (например, C++) конструктор имеет имя, идентичное имени класса, объекты которого он создает, и служит функцией-компонентом класса, конструктором которого он является. Мы же будем придерживаться механизма, предложенного фирмой CA. Вспомним, каким образом создаются стандартные объекты CLIPPER. Для создания, например, объекта класса GET мы вызываем конструктор GetNew(), объект класса ERROR создается конструктором ErrorNew() и так далее. Легко заметить, что имя конструктора состоит из двух составляющих: имени класса и контекста NEW, определяющего создание нового объекта.

Теперь попробуем представить синтаксис объявления конструктора объекта для произвольного класса:

```
CLASS <Имя_класса>                                    ;
CONSTRUCTOR                                           ;
      <Имя_конструктора>([<Список_параметров>] )
   ...
   // Инициализация компонентов объекта
   ...
END CONST[RUCTOR]
Здесь <Имя_класса>          — имя класса, для которого определяется
                              конструктор;
      <Имя_конструктора>    — имя функции, реализующей действия
                              конструктора;
      <Список_параметров>   — список параметров,
                              передаваемых
                              конструктору.
```

Здесь возникает вполне понятный вопрос: почему в объявлении имени конструктора мы указали аргумент <Имя_конструктора>? Ведь конструктор имеет строго определенное имя — <Имя_класса>NEW(). Ответ прост: этим мы расширяем возможности по созданию объектов класса. Определимся, объект класса может быть создан несколькими различными конструкторами, но только имеющий словосочетание <Имя_класса>NEW() будет считаться основным конструктором, а остальные вспомогательными. Внимание: класс должен всегда иметь хотя бы один основной конструктор!

Но наш конструктор не является методом класса, объекты которого он создает. Каким же образом он будет осуществлять доступ к данным (компонентам), скрытым внутри тела объекта? Для этого определим еще одну псевдопеременную, которая возвращает указатель на тело вновь создаваемого объекта класса:

```
#xtranslate this => __oObjBody
```

Переменная __oObjBody автоматически создается при возникновении объекта класса и инициализируется указателем на тело этого объекта.

Пример 1 показывает объявление и создание класса POINT, который описывает координаты точки на экране и определяет ряд методов для манипулирования этой точкой.

Вот теперь мы, имея простейший аппарат объявления классов (листинг 1) и создания объектов, дополним его свойствами ООП. И начнем, пожалуй, с одного из самых важных — с механизма наследования.

Наследование

Наследование является одним из важных, если не самым главным свойством, выделяющим ООЯП. Наследование — механизм получения данных и методов от объекта "родителя" ("предка"). На основе наследования классы должны предусматривать создание расширенной системы предопределенных типов. Механизм наследования нам и предстоит разработать, но для полного понимания функционирования этого механизма рассмотрим понятие иерархии классов или "дерева наследования".

Понятие иерархии классов

Любой из ООЯП позволяет программисту создавать новые классы как производные от уже существующих. Такой производный класс ("потомок") наследует структуру данных и методы родительского класса и дополняет их новыми данными и методами. Таким образом каждый класс входит в одну иерархию классов (дерево классов или наследования). В вершине любой иерархии классов находится один или несколько базовых (абстрактных) классов. Компоненты абстрактного класса могут наследоваться производными классами, но экземпляры объектов абстрактного класса не могут быть созданы!

Наследование свойств базовых классов

Каким образом должен функционировать механизм получения данных и методов от объекта предка? Однозначно можно сказать, что и описатель класса, и его конструкторы должны обладать возможностью обращаться за наследуемой информацией к своему предку. Тогда необходимо в наши команды добавить ряд дополнительных возможностей.

1. Для построения иерархии классов создаваемый класс должен обращаться к своему предку, предположим, следующим образом:

```
CLASS <Имя_класса> <nAccess:PUBLIC,PRIVATE>
<Базовый_класс>
   ...
   // Описание компонентов класса
   ...
END CLASS
```

Таким образом, базовый класс (предок) передаст все наследуемые данные и методы производному классу по модификатору доступа nAccess.

2. Генерируя объект на основе иерархии классов, конструктор создаваемого класса должен вызвать конструктор базового класса для инициализации объекта базового класса:

```
CLASS <Имя_класса>                                    ;
CONSTRUCTOR                                           ;
      <Имя_конструктора>([<Список_параметров_1>]);
BASE [CONSTRUCTOR]                                    ;
      <Имя_базового_конструктора>                     ;
                        ([<Список_параметров_2>])
   ...
   // Инициализация компонентов объекта
   ...
END CONST[RUCTOR]
Здесь <Имя_базового_конструктора>  — имя функции,
                                     реализующей
                                     действия
                                     базового
                                     конструктора;
```

Ключевое слово CONSTRUCTOR в словосочетании BASE CONSTRUCTOR можно опускать для простоты использования.

Рассмотрев механизмы наследования ООЯП, например, C++, можно заметить, что модификатор доступа Access в объявлении класса может принимать различные значения, например, PRIVATE и PUBLIC. Этот модификатор доступа определяет видимость наследуемых данных и методов для потомка базового класса. Имея достаточный опыт программирования на C++, хочу заметить, что хотя эти возможности и предоставляют богатый набор действий по созданию иерархии, на практике обычно наталкиваешься на трудности при разборе таких деревьев, в которых ограничены области видимости компонентов классов. Поэтому первоначально можно отказаться от предоставления таких возможностей по скрытию данных и методов классов "предков".

Теперь рассмотрим создание вершины иерархии классов, а именно набора абстрактных базовых классов. В принципе, абстрактный класс ничем не отличается от обычных классов. Он имеет идентичные описатели классов и свой конструктор, но составляющие его компоненты (данные и методы) всегда абстрактны, так как набор его методов и данных может быть переопределен в производных классах. То есть в словаре методов абстрактного класса некоторые его ме-

Пример 1

```
// Начало файла Point.prg
#include"CGeneric.ch"          // Здесь определяется синтаксис новых
                               // команд языка, служащих для создания
                               // классов (см. листинг 1).

// Это объявление класса POINT ("точка")
CLASS Point
  // Члены класса :
  INSTANCE nX                  // горизонтальная координата(колонка),
  INSTANCE nY                  // вертикальная координата (строка)
  // Методы класса :
  METHOD Grow( x,y )           // Изменение координат точки.
  METHOD Check( oPoint )       // Сравнение двух точек на экране.
  //
END CLASS

// ----- Конструкторы ----------------------------------------

// PointNew() - основной конструктор класса POINT создает
// объект класса с определением координат
// точки на экране монитора.
CLASS Point CONSTRUCTOR PointNew()
  this:nX := 0
  this:nY := 0
END CONSTRUCTOR

// SetPoint()   —   дополнительный конструктор класса POINT
//                  создает объект класса без определения
//                  координат точки на экране монитора.
CLASS Point CONSTRUCTOR SetPoint( x,y )
  this:nX := if( x != NIL .and. x >= 1 .and. x <= 24 ,x ,0 )
  this:nY := if( y != NIL .and. y >= 1 .and. y <= 80 ,y ,0 )
END CONSTRUCTOR

// ------ Методы ---------------------------------------------

// Назначение :
//      Grow()  —   изменение координат точки на экране в
//                  указанном направлении. Направление изменения
//                  определяется знаком параметров, передаваемых
//                  методу.
// Параметры :
//      x       —   указывает значение, на которое изменяется
//                  координата nX (колонка), и знак - направление
//                  изменения значения координаты.
//      y       —   указывает значение , на которое изменяется
//                  координата nY (строка), и знак - направление
//                  изменения значения координаты.
// Возвращаемое значение: нет.
STATIC FUNCTION Grow(x,y)
  ::nX += if( x == NIL, 0, x)
  ::nY += if( y == NIL, 0, y)
RETURN

// Назначение:
//      Check()  —   сравнение двух экземпляров объекта POINT.
// Параметры:
//      oPoint   —   указывает на объект класса POINT, с которым
//                   требуется провести сравнение объектов.
// Возвращаемое значение:
//      -1       —   координаты нашего объекта меньше указанного.
//      0        —   объекты адресуют одну и ту же точку на экране.
//      1        —   координаты нашего объекта больше указанного.
STATIC FUNCTION Check( oPoint )
RETURN   if( ::nX > oPoint:nX .and. ::nY > oPoint:nY, 1, ;
  if( ::nX == oPoint:nX .and. ::nY == oPoint:nY, 0, -1 ) )

// Конец файла Point.prg
```

■

тоды могут быть замещены на новые методы из класса потомка. Для лучшего понимания действий, выполняемых производными классами при наследовании свойств базового класса, введем понятие виртуального метода. Сразу отмечу, что наше понятие виртуального метода несколько отличается от классического понимания в ООЯП.

Виртуальный метод — это метод, действия которого могут быть переопределены, заменены на действия, зависящие от конкретной реализации объекта, созданного на основе класса.

Приведем синтаксис объявления виртуального метода:

```
VIRTUAL METHOD <Имя_функции>                    ;
                    ([<Список_параметров>])
```

Ключевое слово VIRTUAL указывает на то, что данный метод класса должен быть переопределен в производных классах. Но функция, реализующая данный метод, все равно должна быть определена в абстрактном классе для предотвращения случайных вызовов из производных классов.

Множественное наследование

Множественное наследование — это возможность производного класса наследовать данные и методы нескольких родительских классов. В разрабатываемом нами языке этот механизм выглядел бы следующим образом:

```
CLASS    <Имя_класса>                                ;
         <nAccess>                                   ;
         <Базовый_класс1>,...,<Базовый_классN>
   ...
   // Описание компонентов класса
   ...
END CLASS
```

Из примера видно, что класс <Имя_класса> наследует методы и данные нескольких родительских классов. Конструктор этого класса выглядел бы следующим образом:

```
CLASS <Имя_класса>                                   ;
CONSTRUCTOR                                          ;
      <Имя_конструктора>([<Список_параметров_1>]);
BASE                                                 ;
      <Имя_базового конструктора1>                   ;
                    ([<Список_параметров_2>]);
                    ,...,                            ;
      <Имя_базового конструктораN>                   ;
                    ([<Список_параметров_N+1>])
   ...
   // Инициализация компонентов объекта
   ...
END CONST[RUCTOR]
```

Множественное наследование применяется при создании классов, которые используются не сами по себе, а для того, чтобы изменить поведение какого-то другого класса.

Пример 2 показывает объявление и создание небольшой иерархии классов на основе абстрактного класса OBJ, но без использования механизма множественного наследования.

Пример 2

```
// Начало файла Obj.prg
#include"CGeneric.ch"     // Здесь определяется синтаксис новых
                          // команд языка, служащих для создания
                          // классов (см. листинг 1).

// Это объявление абстрактного класса OBJ.
CLASS Obj
  // Члены класса :

  // Методы класса :
  VIRTUAL METHOD shutDown()   // Абстрактный деструктор
  //
END CLASS

// ----- Конструкторы ----------------------------------------

// ObjNew()     —    основной конструктор класса OBJ
//                   создает объект класса.
CLASS Obj CONSTRUCTOR ObjNew()
END CONSTRUCTOR

// Назначение:
//      shutDown()   —    абстрактный деструктор объектов,
//                        производных от класса OBJ.
// Параметры: нет.
// Возвращаемое значение: нет.
STATIC FUNCTION shutDown()
RETURN

// Конец файла Obj.prg



// Начало файла Point.prg
#include"CGeneric.ch"

// Это объявление класса POINT ("точка"),
// производного от абстрактного класса OBJ.
CLASS Point PUBLIC Obj
  // Члены класса:
  INSTANCE nX                 // горизонтальная координата(колонка)
  INSTANCE nY                 // вертикальная координата (строка )
  // Методы класса:
  // POINT не переопределяет метод shutDown() класса OBJ,
  // POINT просто наследует его.
  METHOD Grow( x,y )          // Изменение координат точки.
  METHOD Check( oPoint )      // Сравнение двух точек на экране.
  //
END CLASS

// ----- Конструкторы ----------------------------------------

// PointNew()   —    основной конструктор класса POINT
//                   при создании объекта класса POINT
//                   вызывает конструктор базового класса OBJ.
CLASS Point CONSTRUCTOR PointNew() BASE ObjNew()
  this:nX := 0
  this:nY := 0
END CONSTRUCTOR

// SetPoint()   —    дополнительный конструктор класса POINT
//                   при создании объекта класса POINT
//                   вызывает конструктор базового класса OBJ.
CLASS Point CONSTRUCTOR SetPoint( x,y ) BASE CONSTRUCTOR ObjNew()
  this:nX := if( x != NIL .and. x >= 1 .and. x <= 24 ,x ,0 )
  this:nY := if( y != NIL .and. y >= 1 .and. y <= 80 ,y ,0 )
END CONSTRUCTOR

// Описания методов остались без изменений!
....
....

// Конец файла Point.prg
```

Инкапсуляция данных и полиморфизм

Итак, дорогой читатель, мы с вами подошли к теме инкапсуляции данных. Как нами было уже определено, инкапсуляция — это механизм скрытия данных (компонентов) в теле объекта для доступа из внешних модулей. Мы также определили, что внутренние данные объекта могут быть скрыты от классов потомков, наследующих компоненты и функции-компоненты родительских классов. С последним положением мы уже определились: все компоненты родительских классов будут иметь модификатор доступа PUBLIC для простоты использования. Ну а с первым положением о скрытии данных от модулей извне нам и предстоит сейчас разобраться.

Пример 2 (продолжение)

```
// Начало файла Pixel.prg
#include"CGeneric.ch"

// Это объявление класса PIXEL, производного от класса POINT.
// Класс PIXEL характеризует объект изображения в цвете
// и с координатами на экране монитора.
CLASS Pixel PUBLIC Point
  // Члены класса:
  INSTANCE color := SetColor()    // цвет изображения,
  INSTANCE oldColor               // цвет до создания объекта,
  INSTANCE Attrib := .F.          // атрибут (видим/скрыт).
  // Методы класса:
  VIRTUAL METHOD Draw()           // отображение текущего объекта,
          METHOD shutDown()       // переопределение деструктора.
          // Изменение цвета.
          METHOD setColor( cColor ) TO _xsetColor
          METHOD getColor()       // Определяет текущий цвет.
  //
END CLASS

// ----- Конструкторы ----------------------------------------

// PixelNew(x,y)      -    основной конструктор класса PIXEL
//                         при создании объекта класса PIXEL
//                         вызывает конструктор базового класса POINT.
CLASS Pixel CONSTRUCTOR PixelNew(x,y,cColor) BASE SetPoint(x,y)
  // Класс PIXEL наследует данные и методы, скрытые в теле
  // объекта базовых классов POINT и OBJ.
  this:oldColor := SetColor()
  this:setColor( cColor )         // Установка цвета объекта.
  this:color := this:getColor()
END CONSTRUCTOR

// ------ Методы ---------------------------------------------

// Назначение:
//      Draw() - виртуальный метод отображения текущего объекта.
// Параметры: нет.
// Возвращаемое значение: нет.
STATIC FUNCTION Draw()
RETURN

// Назначение:
//      shutDown() - деструктор объекта PIXEL.
// Параметры: нет.
// Возвращаемое значение: нет.
STATIC FUNCTION shutDown()
  // Восстановление цвета изображения.
  ::setColor( ::oldColor )
  ::Attrib := .F.             // Установить атрибут в положение "скрыт".
RETURN

// Назначение:
//      SetColor() - изменение цвета изображения.
// Параметры:
//      cColor - новое значение цвета.
// Возвращаемое значение: нет.
STATIC FUNCTION _xsetColor( cColor )
  if( cColor != NIL .and. VaType(cColor) == "C" )
      ::color := cColor    // новый цвет
      SetColor( ::color )  // изменение цвета изображения
  end
RETURN

// Назначение:
//      getColor() - читает текущий цвет объекта.
// Параметры: нет.
// Возвращаемое значение:
//      - текущий цвет объекта (в формате SetColor()).
STATIC FUNCTION getColor()
RETURN ::color

// Конец файла Pixel.prg
```

Предположим, каждый компонент класса может иметь один из трех возможных атрибутов: PUBLIC — общедоступный, PROTECTED — защищенный и EVENT — обрабатывающий событие. Рассмотрим последовательно каждый из этих атрибутов.

1. Общедоступный (PUBLIC)

Компонент public доступен для изменений из любых точек программы, где виден объект указанного класса, для которого компонент определен. По умолчанию все компоненты класса имеют атрибут PUBLIC.

Приведем синтаксис объявления общедоступного компонента:

```
INSTANCE nColor
```

2. Защищенный (PROTECTED)

Защищенный компонент доступен для изменения только внутри тела объекта, в котором он определен, а также внутри конструктора указанного класса. Область видимости этого компонента ограничена областью видимости объекта, как и у компонента с атрибутом PUBLIC.

Приведем синтаксис объявления защищенного компонента:

```
INSTANCE nMode PROTECTED
```

3. Обрабатывающий событие (EVENT)

Предположим, изменение значения компонента невозможно без реорганизации внутренних данных объекта или любая попытка изменения значения этой переменной может быть выполнена при определенных условиях. Для таких переменных введем атрибут EVENT — обрабатывающий событие. Компонент с этим атрибутом доступен для изменений из любых точек программы, где виден объект указанного класса, для которого он определен, при этом управление программой будет передано обработчику события, указанному для компонента.

Приведем синтаксис объявления компонента, обрабатывающего событие:

```
INSTANCE nMode EVENT ModeControl
Здесь ModeControl — функция-обработчик события.
```

Предположим, что nMode — это режим открытия потока ввода-вывода, следовательно, любые изменения режима должны быть проанализированы потоком.

```
STATIC FUNCTION ModeControl( Mode )
RETURN if( ::Opened, ::nMode, ::SelfVar(::nMode,Mode) )
Opened   — хранит информацию о том, открыт в данный
           момент поток или нет;
nMode    — режим открытия потока;
Mode     — новое значение режима.
```

Вы, несомненно, обратили внимание на функцию-компонент SelfVar(). Это псевдометод, определенный для всех классов, создаваемых приложением. Он предназначен для доступа к телу объекта в обход механизма доступа. Он позволяет напрямую изменять значения компонентов класса, включая EVENT и PROTECTED переменные. Этот псевдометод имеет доступ к данным объекта, будучи вызванным только из экспортируемых методов класса или его конструктора. Вызов его не отличается от вызова обычного метода. SelfVar() принимает два значения в качестве параметров: первый — это компонент, значение которого требуется изменить, и второй — новое значение компонента. Вызов SelfVar() в нашем примере необходим, чтобы избежать бесконечного рекурсивного вызова ModeControl().

Теперь перейдем к рассмотрению последнего из основных свойств ООЯП — полиморфизма. Полиморфизм придает ООЯП некоторое подобие интеллекта. Тот факт, что класс может содержать более одного метода и, следовательно, отвечать на сообщения нескольких типов, свидетельствует о полиморфизме. Кроме этого, функциям CLIPPER при передаче параметров безразлично, какие именно типы данных они передают, то есть пользователь сам может определить тип данных и количество передаваемых параметров.

Например, мы можем определить функцию-компонент Format(), определяющую тип принимаемого

значения nVal и функцию-компонент Print(), выдающую форматную строку на экран:

```
METHOD Format( nVal )
METHOD Print( cString )
```

Контекст объявления функции-компонента Format() выглядит следующим образом:

```
STATIC FUNCTION Format( nVal )
   // Определяем тип параметра nVal.
      local cType := ValType(nVal)
   // Определив тип передаваемого параметра,
   // мы можем описать и действия,
   // которые необходимо произвести с этим параметром.
   Do Case
      Case cType == "A"
      // Распечатка массивов
      AEval( nVal, { |val| ::Format(val) } )
      Case cType == "D"
      // Распечатка данных
      ::Print( "Type data : " + DTOC( nVal ) )
      Case cType == "L"
      // Распечатка логического значения
      ::Print( "Type logical : " + if( nVal, ".T.",".F.") )
      //
      // и так далее
      ...
      //
      Case cType == "N"
      // Распечатка чисел
      ::Print( "Type numeric : " + STR( nVal ) )
   EndCase
RETURN
```

И последнее, что хотелось бы заметить. К сожалению, одно из свойств полиморфизма нам пока не удалось реализовать в созданном ООЯП — это замещение операторов. Можно, конечно, создать какое-то подобие, но лучшая возможность — внедрение такого механизма в исполнительную систему CLIPPER.

Пример 3 дополняет пример 2 командами скрытия данных внутри объектов.

Обратите внимание, что при изменении цвета объекта необходима перерисовка объекта на экране новым цветом. Для этого определим функцию-обработчик события и установим атрибут внутренней переменной COLOR в EVENT. Обработчик события ColorCtrl() определяет, видим ли объект в текущий момент, и если да, то перерисовывает его на экране монитора.

Теперь на основе класса PIXEL мы строим производный класс VIEW, который наследует свойства класса PIXEL и определяет свои компоненты (пример 4).

Этот пример наглядно иллюстрирует все возможности нашего механизма создания и манипулирования данными типа "объект". Этот пример нагляден, но на практике вряд ли будет необходимость в создании такого экземпляра объекта класса VIEW, каким он реализован в нашем примере. Скорее для отображаемых объектов класс VIEW тоже будет абстрактным и более богатым набором методов для манипулирования внутренними данными объектов.

Пример 3

```
// Начало файла Obj.prg
#include"CGeneric.ch"     // Здесь определяется синтаксис новых
                          // команд языка, служащих для создания
                          // классов (см. листинг 1).

// Это объявление абстрактного класса OBJ.
CLASS Obj
  // Члены класса :

  // Методы класса :
  VIRTUAL METHOD shutDown()   // абстрактный деструктор
  //
END CLASS

// Описания конструктора и методов остались без изменений!
....
....

// Конец файла Obj.prg



// Начало файла Point.prg
#include"CGeneric.ch"

// Это объявление абстрактного класса POINT ("точка"),
// производного от абстрактного класса OBJ.
CLASS Point PUBLIC Obj
  // Члены класса:
  // для координат точки установлен атрибут
  // "защищенный от изменения", изменения координат возможны
  // только через метод Grow().
  INSTANCE nX PROTECTED       // Горизонтальная координата(колонка).
  INSTANCE nY PROTECTED       // Вертикальная координата (строка).
  // Методы класса:
  // POINT не переопределяет метод shutDown() класса OBJ,
  // POINT просто наследует его.
  METHOD Grow( x,y )          // Изменение координат точки.
  METHOD Check( oPoint )      // Сравнение двух точек на экране.
  //
END CLASS

// ----- Конструкторы ----------------------------------------

// PointNew()  -   основной конструктор класса POINT,
//                 при создании объекта класса POINT
//                 вызывается конструктор базового класса OBJ.
CLASS Point CONSTRUCTOR PointNew() BASE ObjNew()
  // Псевдометод SelfVar() устанавливает значения защищенных
  // компонент класса POINT непосредственно в теле объекта,
  // минуя при этом механизм доступа к компонентам класса.
  this:SelfVar( this:nX,0 )
  this:SelfVar( this:nY,0 )
END CONSTRUCTOR

// SetPoint()  -   дополнительный конструктор класса POINT,
//                 при создании объекта класса POINT
//                 вызывается конструктор базового класса OBJ.
CLASS Point CONSTRUCTOR SetPoint( x,y ) BASE CONSTRUCTOR ObjNew()
  this:SelfVar( this:nX,if(x! = NIL.and.x > = 1.and.x < = 24,x,0) )
  this:SelfVar( this:nY,if(y! = NIL.and.y > = 1.and.y < = 80,y,0) )
END CONSTRUCTOR

// ------ Методы ---------------------------------------------

// Назначение:
//     Grow()  -   изменение координат точки на экране
//                 в указанном направлении. Направление
//                 изменения определяется знаком параметров,
//                 передаваемых методу.
```

Что же дальше?

Такой вопрос вполне может прозвучать в завершении этой статьи. Тем более, что на сегодня существует несколько подобных интерфейсов связи со списками классов. Приведу некоторые из них.

1. Одним из первых был создан интерфейс, разработанный Питером Фризом (фирма CyberSoft), реализованный в библиотеке OCLIP. Библиотека включает в себя несколько модулей по объявлению классов и созданию простейших иерархий классов. Эту библиотеку можно получить на многих станциях BBS.

2. Почти параллельно с ней была создана библиотека HOOP, разработанная в учебном центре "SMART+" А.В. Шестерниным. Библиотека включает в себя не только модули поддержки интерфейса к списку классов, но и развитый язык объявления пользовательских классов. Кроме того, в библиотеку включен небольшой на-

Пример 3 (продолжение)

```
// Параметры:
//      x     —     указывает значение, на которое изменяется
//                  координата nX (колонка), и знак - направление
//                  изменения значения координаты.
//      y     —     указывает значение, на которое перемещается
//                  координата nY (строка), и знак - направление
//                  изменения значения координаты.
// Возвращаемое значение: нет.
STATIC FUNCTION Grow(x,y)
  ::SelfVar( ::nX, ::nX + if( x = = NIL , 0 , x ) )
  ::SelfVar( ::nY, ::nX + if( y = = NIL , 0 , y ) )
RETURN

// Описание метода Check() осталось без изменений!
....

// Конец файла Point.prg



// Начало файла Pixel.prg
#include"CGeneric.ch"

// Это объявление абстрактного класса PIXEL, производного от
// класса POINT.
// Класс PIXEL характеризует объект изображения в цвете и с
// координатами на экране монитора.
CLASS Pixel PUBLIC Point
  // Члены класса :
  INSTANCE color EVENT ColorCtrl      // цвет изображения,
  INSTANCE oldColor                   // цвет до создания объекта,
  INSTANCE Attrib: = .F. PROTECTED    // атрибут (видим/скрыт).
  // Методы класса :
  VIRTUAL METHOD Draw()               // Отображения текущего объекта.
      METHOD shutDown()               // Переопределение деструктора.
      // Изменение цвета.
      METHOD setColor( cColor ) TO _xsetColor
      METHOD getColor()               // Определяет текущий цвет.

  //
END CLASS

// ----- Конструкторы ----------------------------------------

// Описания конструктора и методов остались без изменений,
// кроме метода shutDown(). Так как переменная Attrib имеет
// атрибут PROTECTED, в методе shutDown() необходимо
// модифицировать строку присвоения значения этой переменной
// на: ::SelfVar(::Attrib,!::Attrib).

....
....

// Назначение:
//      ColorCtrl() - обработчик события изменения цвета объекта.
// Параметры:
//      cColor - устанавливаемое значение цвета.
// Возвращаемое значение: нет.
STATIC FUNCTION ColorCtrl( cColor )
  if( cColor ! = NIL .and. ValType(cColor) = = "C" )
      ::SelfVar(::color,cColor)       // Новый цвет.
      ::SetColor( ::color )           // Изменение цвета изображения.
      // Если объект видим на экране...
      if( ::Attrib )
          // ...перечертить его.
          ::Draw()
      end
      ::SetColor( ::oldColor )        // Восстановление цвета
  end
RETURN

// Конец файла Pixel.prg
```

Пример 4

```
// Начало файла View.prg
#include"CGeneric.ch"

// Это объявление класса VIEW, производного от класса PIXEL.
// Класс VIEW хранит визуализируемый объект изображения в цвете
// и с координатами на экране монитора.
CLASS View PUBLIC Pixel
  // Члены класса:
  INSTANCE Char          // элемент изображения.
  // Методы класса:
  // переопределение метода DRAW() класса "предка".
  METHOD Draw()          // Отображает Char
  //
END CLASS

// ----- Конструкторы ----------------------------------------

// ViewNew(x,y,sh,color) —    основной конструктор класса VIEW,
//                            при создании объекта класса VIEW
//                            вызывается конструктор базового
//                            класса PIXEL.
CLASS View CONSTRUCTOR ViewNew(x,y,ch,color) BASE PixelNew(x,y,color)
  this:Char : = ch
END CONSTRUCTOR



// ------ Методы ---------------------------------------------

// Назначение:
//      Draw() - отображение текущего объекта.
// Параметры: нет.
// Возвращаемое значение: нет.
STATIC FUNCTION Draw()
  if( !::Attrib )
      ::SelfVar( ::Attrib,!::Attrib )
  end
  @@ ::nY,::nX SAY ::Char
RETURN

// Конец файла View.prg

// Начало файла Main.prg
//
// Функция строит объект класса View и отображает его на экране.
//
FUNCTION Main()
  // Создание экземпляра объекта VIEW.
  local String : = ViewNew(10,20,"Hello, World !!!","R/B")
  // Отображение объекта.
  String:draw()
  ....

  // Изменение цвета объекта и автоматическая перерисовка
  // объекта String на экране монитора.
  String:color : = "+W/BG,R/B,,,,W/N"
  ....

  // Удаление объекта.
  String:shutDown()
RETURN

// Конец файла Main.prg
```

бор пользовательских классов и другие вспомогательные процедуры. Но при развитых средствах инкапсуляции данных полностью отсутствует механизм наследования данных в производных классах.

Кроме этого, как уже отмечалось ранее, в статье А.М. Рытова имеются примеры получения пользовательских классов на основе недокументированных функций фирмы CA и создан простейший аппарат объявления классов внутри пользовательских функций.

Почти все перечисленные библиотеки имеют минимальный набор средств только для объявления классов пользователем и создания объектов на основе этих классов. Но мы не будем ограничиваться только интерфейсом к классам. Разработанный аппарат, конечно, можно применять на практике и создавать свои классы, иерархии классов и библиотеки классов, хотя оптимальным является использование уже готового набора самых необходимых инструментов. Набор таких инструментов и предоставляет библиотека Visual Clipper. В состав библиотеки входит механизм создания классов пользователя, рассмотренный нами в этой статье, и библиотека классов, построенная на основе описанного интерфейса.

Библиотека Visual Clipper является средством написания прикладных программ, работающих в среде DOS, реализующим новый взгляд на проектирование программного обеспечения. Библиотека может быть использована для создания многофункционального интерфейса взаимодействия, включающего окна, диалоговые панели, меню, поддержку манипулятора "мышь" и многое другое. В состав библиотеки входят более 150 различных классов, включая потоки ввода-вывода, хеш-таблицы, стек и многое другое. Кроме того, в библиотеку входит большое количество вспомогательных функций, включая процедуры фоновой печати, работу с дисками и многие другие.

Библиотека Visual Clipper создана по аналогии с известным пакетом Turbo Vision фирмы Borland и предоставляет пользователю многооконную оболочку, управляемую событиями, с использованием языка CLIPPER версии 5.0 и выше. От вас не требуется мгновенного знания основ объектно-ориентированного программирования, применяйте уже готовые конструкции.

Если Вы будете использовать библиотеку Visual Clipper и объектно-ориентированное программирование, Вам не придется изобретать велосипед: Вы сможете просто его унаследовать!

Заключение

Отметим, что с ростом числа специалистов, отдающих предпочтение методам объектно-ориентированного программирования, можно предположить, что в недалекой перспективе программисты будут компоновать свои программы из более крупных "строительных" блоков, чем функции и процедуры, и использовать именно стандартные объекты. И следовательно, накопление опыта объектно-ориентированного программирования Вам необходимо уже сейчас!

Сетевой адрес автора: FidoNet 2:5020/35.18 (Станция Telesoft BBS работает с 20 до 9 часов. Телефон: (095) 945-50-56.) ■

Листинг 1

```
// Начало файла CGeneric.ch

/*********************
* Назначение:  Файл предназначен для генерации классов
*              пользователя и включает в себя описание
*              набора команд для создания динамических(!)
*              классов.
*********************/

#ifndef _CGENERIC_CH
#define _CGENERIC_CH

// Доступ сообщений к телу объекта, псевдопеременные.
#xtranslate This = > __oObjBody          // Указатель на вновь созданный
                                         // объект.
#xtranslate Self = > _qSelf()            // Возвращает тело объекта.
#xtranslate :: = > _qSelf():             // Возвращает тело объекта.

// Доступ сообщений к защищенным компонентам объекта,
// псевдометоды.
// Доступ из экспортируемых методов.
#command _qSelf():SelfVar( _qSelf():<pp> ,<nn> ) = >              ;
         _SetVars( _qSelf(),<"pp">,<nn>,.t. )
// Доступ из конструкторов объектов.
#command __oObjBody:SelfVar( __oObjBody:<pp> ,<nn> ) = >          ;
         _SetVars( __oObjBody,<"pp">,<nn>,.f. )

// !!! К Л А С С Ы !!!

// Начало описания класса.
// Команда создает новый класс, определяет члены и методы класса.
#command CLASS     <class> [<nAccess:PUBLIC,PRIVATE>]             ;
       <bclass,...> ] = >                ;// Объявление класса.
FUNCTION <class> ()                      ;// Имя функции описателя класса.
LOCAL aClass                             ;// Указатель на описатель.
LOCAL BasedClass [:= __CrBClass({ <(bclass)> } ]    ;// Базовый класс.
aClass := StructClass( <(class)> ,  ;// Создание класса.
if( BasedClass != NIL, BasedClass, {{"",{},{},{},{},{} } ),          ;
.nAccess. )

//--------- Варианты команды добавления переменной ---------
// Простейший
#command INSTANCE <msg> [<chk:PROTECTED>] = >                     ;
_AddMember( aClass, <(msg)>,, <.chk.> )
// С присвоением начального значения
#command INSTANCE <msg> := <arg> [<chk:PROTECTED>] = > ;
```

Листинг 1 (продолжение)

```
_AddMember( aClass, <(msg)>, <arg>, <.chk.> )
// С обработкой событий
#command INSTANCE <msg> EVENT <event> = >                         ;
_AddMember( aClass, <(msg)>,, {|val| <event>(val)} )
// Все сразу
#command INSTANCE <msg> := <arg> EVENT <event> = >  ;
_AddMember( aClass, <(msg)>, <arg>, {|val| <event>(val)} )

//--------- Варианты команды добавления метода ---------
// Простейший
#command METHOD <msg> ([<list,...>]) = >                          ;
_AddMethod( aClass, #<msg>, {|<list>| <msg>(<list>)} )
// Когда нельзя использовать такое имя функции
#command METHOD <msg> ([<list,...>]) TO <func> = >                ;
_AddMethod( aClass, #<msg>, {|<list>| <func>(<list>)} )
// Для описания виртуальных методов класса
#translate VIRTUAL METHOD = > METHOD

// Конец описания класса
#command END CLASS = > Return aClass

// !!! Конструкторы классов !!!

// Начало описания конструктора класса.
// Команда регистрирует класс (иерархию классов), созданный
// командой CLASS, и возвращает указатель на вновь созданный объект.
#command CLASS <class> CONSTRUCTOR <func> ;
      [BASE [CONSTRUCTOR] <bfunc,...> ] = > ;
FUNCTION <func>                          ;// Имя конструктора.
STATIC nClassId                          ;// Код класса и описатель.
STATIC __oObjBody                        ;// Указатель на тело объекта.
PUBLIC __ptrObjClass := CreateClass( <class>(),@nClassId)         ;
__oObjBody := __ptrObjClass              ;// Указатель на тело объекта.
__CrHierar() [; Eval({|| <bfunc>()}) ]   ;// Конструктор базового класса.
__ClrHiera()                             // Очистка иерархии классов.

// Здесь идет "тело" функции конструктора указаного класса...

// Конец описания конструктора
// (конструктор возвращает готовый объект).
#command END CONSTRUCTOR    = > RETURN __oObjBody
#command END CONST          = > RETURN __oObjBody

#endif
// Конец файла CGeneric.ch
```

■

Колледж информационных технологий ЭЛЬНУР является первым в России высшим учебным заведением, осуществляющим профессиональную подготовку пользователей персональных компьютеров и менеджеров в области вычислительной техники.

Колледж является негосударственным (частным) образовательным учреждением и имеет лицензию Московского Департамента образования на ведение профессиональной образовательной деятельности. Колледж организован совместно с Московским издательско-полиграфическим колледжем им. Ивана Федорова.

Колледж создан при поддержке и активном участии фирм:

IBM — в лице постоянного представительства IBM в Москве;

NeXT — в лице фирмы "ТАЛУС", авторизованного дистрибьютера фирмы NeXT;

Apple — в лице генерального дистрибьютера — фирмы "RUI".

В колледже открыто три факультета: компьютерной графики, компьютеризации издательской деятельности, компьютеризации финансовой деятельности.

Факультет компьютерной графики первым в России будет осуществлять подготовку специалистов для телевидения, видеостудий, рекламных агентств и издательств.

Факультет компьютеризации издательской деятельности будет готовить специалистов в области компьютерной подготовки изданий.

Факультет компьютеризации финансовой деятельности будет готовить специалистов в области систем, осуществляющих компьютерную обработку экономической и финансовой информации, включая банковские компьютерные системы.

При колледже действуют курсы повышения квалификации, а также специальное отделение для лиц с высшим образованием.

В колледж принимаются лица, имеющие среднее образование, а на специализированное отделение — высшее образование. Зачисление в колледж осуществляется по результатам собеседования.

Срок обучения на всех факультетах на дневном отделении — 3 года, на вечернем — 4 года, на специализированном отделении — 1 год. Обучение платное.

По окончании колледжа выдается диплом о высшем образовании с присвоением степени бакалавра, оказывается содействие в трудоустройстве.

Уважаемые читатели!

Мы предлагаем Вашему вниманию

компьютерный "ларек".

Мы преследуем две основные цели:

– дать возможность отечественным разработчикам инструментальных средств выйти на российский рынок программного обеспечения;

– обеспечить как можно более широкий круг читателей недорогими и оригинальными средствами разработки программного обеспечения.

Код	Продукт	Краткая характеристика	Объем поставки	Цена, $	Требования к оборудованию	Поддержка	Число продаж
SL-01	Turbo Games	Объектно-ориентированная библиотека для Turbo Pascal 6.0. Спрайты, звуковые эффекты, трехмерная графика, стереоэффекты. Доступ к ZIP файлам.	2 дискеты + документация	12 для индивидуальных покупателей, 25 для организаций	EGA адаптер, IBM PC/AT	+	23
SL-02	Data Vision 1.5	Объектно-ориентированная библиотека для Turbo Pascal 6.0. Создание и управление реляционными базами данных. Поддерживаются DBF и DB форматы, а также электронные таблицы.	1 дискета + документация	25	IBM PC/XT	+	43
SL-03	Atoll	Библиотека для Turbo C. Динамический просмотр и масштабирование графических изображений.	2 дискеты + документация	10	IBM PC XT/AT, EGA/VGA, PCX-system	+	
IS-01	HyperRef	Инструментальное средство для создания гипертекстовых систем. Поддержка текстовой информации, изображений и исполняемых программ.	2 дискеты	70	IBM PC/AT, EGA/VGA, 3 Мб на HDD, 350 Кб ОЗУ — просмотр документа, 450 Кб — редактирование документа.		1
SC-01	QuickTools 1.0	Средство разработки диалога в Paradox.	1 дискета + документация	12	IBM PC/AT, EGA/VGA, 3 Мб на HDD, 1-2 Мб ОЗУ.	+	8
SC-02	QB Tools 3.0	Средство разработки диалога, а также поддержки DBF формата для Quick и Professional Basic.	1 дискета	30 для индивидуальных покупателей, 50 для организаций	IBM PC/AT, EGA, 3 Мб на HDD, 640 Кб ОЗУ	+	
UT-01	ArcView 6.1	Просмотр и редактирование заголовков архивных файлов с возможностью подключения собственных типов архивов.	1 дискета 360 Кб	1	IBM PC/AT, Hercules, EGA/VGA	+	
SE-01	Crypter 2.0	Система шифрования файлов. Поддерживаются алгоритмы DES, ГОСТ.	1 дискета 360 Кб	12	IBM PC/AT, Hercules, EGA/VGA	+	
SC-01	АвтоГРАФ	Пакет для создания на экране графиков с результатами обработки числовых данных и печати их на 9 и 24-игольчатых принтерах. Поддержка форматов ЕСКД.	1 дискета 360 Кб	10	IBM PC/XT, EGA/VGA, 500 Кб на HDD, ОЗУ — 200 Кб	+	73

Если Вас заинтересовали приведенные здесь продукты или Вы хотите получить дополнительную информацию, звоните, пишите, приезжайте.

Оплата принимается только в рублях по текущему курсу.

Адрес для писем: 117419 Москва, а/я 765

Тел.: (095) 179-50-69 (Морев Александр). Тел./факс: (095) 238-99-33, 238-31-89, 238-32-78.

И надпись написал...

А. Тихонов

Разработан растровый графич. редактор для VGA. Лицензионная чистота... Русский текст (BGI) под любым углом...
Диалоговый монитор рынка информационных технологий. КомпьютерУолд-Москва, 1993, № 3 (63)

С аналогичной задачей я столкнулся летом 1991 года, когда в течение недели надо было сделать программу, обеспечивающую вывод русских надписей да еще стандартизированными шрифтами Пр3 и Пр5.

Надо сказать, старый добрый Turbo Pascal и BGI Toolkit фирмы Borland здорово тогда меня выручили, а объявление в КомпьютерУолд, вынесенное в эпиграф, натолкнуло на мысль, что решение задачи вывода текста, подвергнутого произвольным линейным преобразованиям (перенос, изменение масштаба по осям и вращение) будет интересно не только мне.

Можно долго обсуждать, хорош или плох графический интерфейс BGI. Он не торопится при выводе графики, требует сравнительно много оперативной памяти, но является фактическим стандартом для компиляторов фирмы Borland, кроме того, позволяет легко писать программы, автоматически настраивающиеся на имеющийся в ПЭВМ графический адаптер.

В стандартный комплект поставки компиляторов Turbo Pascal и Turbo C до конца 1992 года входило четыре фонта (здесь под фонтами понимается совокупность данных, позволяющих выводить литеры определенного начертания), а в Turbo Pascal версии 7 их включено десять.

Встроенные средства компиляторов обеспечивают работу с векторными фонтами формата BGI, позволяя масштабировать надписи и выводить их в четырех взаимно перпендикулярных направлениях.

Образы символов в векторных фонтах, в том числе и BGI, строятся из отрезков прямых. Это позволяет использовать их для вывода надписей не только на экран, но и графопостроитель или координатограф.

Включенный в BGI Toolkit диалоговый редактор фонтов FE.EXE при известном терпении и минимальных художественных способностях, позволяет изготовлять собственные фонты в том числе и с кириллицей. В этом случае можно рекомендовать книгу [1], содержащую большое число различных фонтов, часть из которых можно реализовать. Кроме этого, в настоящее время ряд московских фирм распространяет как русские BGI фонты, так и редакторы для их создания или модификации.

Насколько мне известно, описание структуры фонтов никогда не публиковались. Во всяком случае, в [2], [3], [4] приводятся результаты реконструкций. Видимо, фирма Borland хотела сохранить свободу при выпуске новых версий своих программных средств.

Обычно векторные фонты в формате BGI хранятся в файлах с расширением CHR. Структура BGI фонта приведена на рис. 1 и состоит из описательной части, относящейся к фонту в целом, а также структур и таблиц с представлениями образов символов.

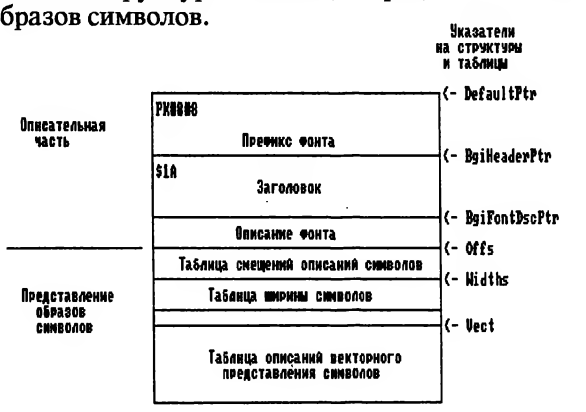


Рис. 1. Структура BGI фонта

Описательная часть не имеет фиксированного размера и состоит из трех структур, представленных в листингах 1-3. Первая из них является

префиксом (листинг 1) и используется для проверки и идентификации фонта. Первое поле FontSignature содержит всегда четыре символа PK#8#8 и используется для проверки корректности фонта, а поля FontDescription и Copyright для идентификации фонта и указания авторских прав на него. Эти поля могут иметь переменную длину, но каждая из них завершается символами #13#10.

Следующая структура — заголовок фонта (BgiFontHeader) приведена в листинге 2 и содержит сигнатуру (байт $1A), размер описательной части (обычно 256 байтов), четырехсимвольное имя, размер фонта без учета описательной части, а также информацию о версии фонта и редакции BGI. Назначение полей структур комментировано в листингах и далее мы будем ссылаться только на те поля, которые необходимы по ходу изложения.

Основные характеристики фонта хранятся в структуре описателя фонта (BGIFontDsc), приведенной в листинге 3. Она, как и структура заголовка, начинается с сигнатуры (символа "+") и содержит число символов в фонте, ASCII код первого символа (для большинства фонтов первым символом является пробел, ASCII код которого равен 32), а также параметры, характеризующие изображения символов фонта (рис. 2).

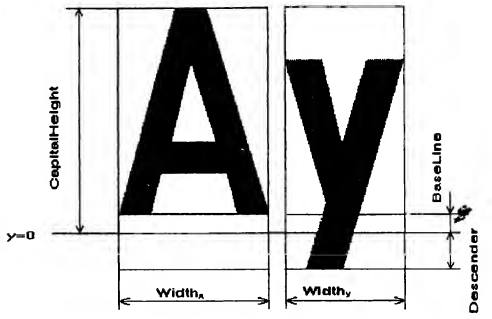


Рис. 2. Структура изображений символов

Изображения литер BGI фонтов выводятся в прямоугольной матрице размером не более 63 по горизонтали и 127 пикселов по вертикали, причем координаты по горизонтали лежат в диапазоне 0...63, а по вертикали -64...+63. Для фонта фиксируется высота строчных букв CapitalHeight, положение базовой линии фонта относительно оси абсцисс (BaseLine) и величина нижнего выноса ("хвостиков" таких символов, как "q" и "y"). Высота прописных букв не фиксирована, а ширина индивидуальна для каждого символа (рис. 2). Структура BGIFontDsc содержит еще несколько полей, но их назначение мы обсудим ниже.

Ширина символов и число векторов, используемых для их изображения, меняются от символа к символу, что приводит к необходимости хранить таблицы смещения ширины символов (WidthTable) и смещения начала представления каждого символа (OffSetTable). Заметим, что таблица WidthTable используется для определения длины строки (рис. 1).

Вывод символа предполагает поиск в таблице смещения начала описания данного символа с последующей выборкой и вычерчиванием векторов изображения до тех пор, пока не встретится признак завершения представления символа.

Рассмотрим теперь, как кодируются изображения символов. Ранее отмечалось, что изображение состоит из векторов, каждый из которых хранится в структуре Vector (листинг 4). Vector содержит координаты x, y конца вектора в формате SHORTINT. Старшие биты координат вектора используются для хранения команд вычерчивания символа.

Нулевое значение старшего бита координаты вектора по оси абсцисс означает команду окончания вывода символа. Вычерчивание векторов, из которых составлено изображение символа, хорошо описывается в терминах команд графопостроителя, а именно: "поднять перо и сдвинуться из точки, где в настоящее время находится перо, на x по оси абсцисс и на y по оси ординат" и "опустить перо и вычертить отрезок прямой с концом в x, y". Так, если установлены старшие биты одновременно в x и y, то необходимо опустить перо и вычертить отрезок прямой, если установлен старший бит в x, а в y сброшен, то перо поднимается и переход осуществляется без вычерчивания. Указанные команды реализуются с помощью процедур MOVETO и LINETO. Некоторое неудобство состоит в том, что, кроме сброса старшего бита, необходимо преобразование полей x, y из SHORTINT в INTEGER.

Перейдем теперь к базовым средствам вывода надписей с помощью BGI фонтов, представленных в листинге 5 и скомпонованных в виде объекта BGIFont.

Конструктор LoadBgiFont осуществляет загрузку в оперативную память фонта с именем FontName, базирование указанных выше структур и таблиц, проверку корректности загруженного фонта. В случае удачной загрузки флажок Loaded становится равным TRUE. Конструктор использует метод SetFontPtrs для настройки адресов структур и таблиц. В Pascal, в отличие от C, отсутствуют механизмы работы с адресной арифметикой. Это вынудило меня использовать фиктивный массив байтов с указателем DefaultPtr.

Установка указателя структуры заголовка фонта (BgiHeaderPtr) осуществляется сканированием памяти в поисках байта с шестнадцатиричным значением $1A.

Структура описания фонта с указателем BgiFontDscPtr смещена относительно начала заголовка на величину, хранящуюся в поле HeaderSize заголовка. Вплотную к структуре описания фонта примыкает таблица смещений символов с указателем Offs. Для хранения смещения требуется два байта, так что размер указанной таблицы равен 2 * SymbolsInFont, где SymbolsInFont — поле в структуре описания фонта с числом символов в данном фонте.

Таблица ширины символов примыкает к таблице смещений, размер ее равен SymbolsInFont.

Наконец, указатель Vect на таблицу с векторными представлениями символов имеет смещение относительно заголовка фонта, хранящееся в поле StrokesOffset структуры описателя фонта.

Метод CheckFont проверяет корректность сигнатур префикса, заголовка и описателя фонта, что гаран-

тирует от неприятных неожиданностей при выводе надписей.

Деструктор Done освобождает память, использованную для работы с фонтом, и сбрасывает флажок Loaded.

Методы SymbolHeight и SymbolFullHeight возвращают высоту строчной буквы и полную высоту с учетом нижнего выноса соответственно.

Метод SymbolWidth возвращает ширину символа C, извлекаемую из таблицы ширины символов.

Центральным в объекте является метод Transform, используемый для преобразования координат и подлежащий замене в объектах — наследниках BGIFont. Изначально Transform осуществляет тождественное преобразование. Заметим также, что при осуществлении преобразований координат в наследниках BGIFont необходимо переписать методы SymbolHeight, FullSymbolHeight, SymbolWidth.

Вывод символа C в точке с координатами x1, y1 в методе OutSymbol начинается с вычисления смещения его векторного изображения (переменная J) и перемещения пера в точку x1, y1. Вывод может осуществляться в двух режимах: в обычном, когда вычерчиваются только контуры символов, и с заполнением, когда после вычерчивания внутренняя часть символа закрашивается с помощью процедуры FILLPOLY. Закраска осуществляется только в том случае, когда установлен флажок FillFlag в структуре описания фонта, кроме того, параметр Fill задан равным TRUE при вызове OutSymbol. Это обусловлено тем, что для закраски необходимо создавать специальные фонты, так чтобы закрашиваемый контур был выведен одним росчерком пера без его подъема. Процедуру FILLPOLY приходится вызывать как при завершении работы символа, так и при подъеме пера, иначе не удается закрасить многосвязные символы, например, "i" или "ы". Поэтому в таких символах, как "O", приходится делать перемычку между внешним и внутренним контуром.

Метод OutString сводится к выводу последовательности символов, входящих в строку S, причем между символами строки оставляется промежуток Gap.

С помощью методов объекта BGIFont выведена надпись, фигурирующая в качестве заголовка статьи (рис. 3).

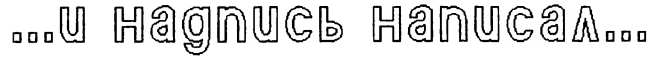


Рис. 3. Пример использования объекта BGIFont

Теперь настала пора решить задачу, поставленную в эпиграфе статьи, — вывести надпись с произвольной ориентацией (листинг 6). Объект Font1 позволяет выводить надписи в любом масштабе по осям координат и с произвольной ориентацией. Для этого, естественно, пришлось переписать метод Transform, а также и SymbolHeight, SymbolFullHeight, SymbolWidth, StringLength. Задание параметров линейного преобразования (масштабы по осям координат MX, MY, угол поворота A в градусах относительно центра X00, Y00) осуществляется методом SetParameters1. Результат использования объекта Font1 представлен на рис. 4.

Рис. 4. Масштабирование и вращение надписи

В качестве еще одного примера модификации объекта BGIFont попробуем вывести надпись по дуге окружности (листинг 7). Параметры дуги устанавливаются методом SetParameters2 объекта Font2. В нем X00, Y00 — центр дуги, а DY — высота дуги относительно горизонтальной линии, проходящей через точку, в которой начинается вывод строки. Данный параметр может принимать как положительные, так и отрицательные значения. Для объекта Font2 пришлось переписать методы Transform и OutString, причем второй метод модифицировался только для того, чтобы растянуть надпись по всей дуге. Результаты представлены на рис. 5. Программа, с помощью которой получены рисунки, приведенные в статье, представлена в листинге 8.

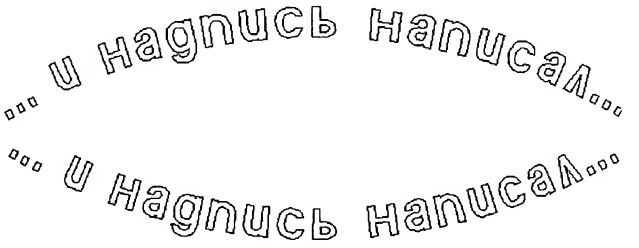


Рис. 5. Вывод надписи по дуге окружности

Естественно, что использование базовых средств вывода надписей не ограничивается приведенными примерами. Так, достаточно просто (хотя реализация несколько громоздка и поэтому здесь не приводится) можно выводить строки вдоль произвольных ломаных линий, а очень красивых эффектов вывода текстов удается достичь, используя при модификации метода Transform различные конформные преобразования... ■

Литература

1. Декоративные шрифты для оформительских работ.– Минск: Полымя, 1987.
2. B. Ezzl. Graphic Programming in Turbo C 2.0. — N.Y.: Addison-Wesley Inc., 1990.
3. W. Schneider. Fonts aus Nachbars Garten//C't Magazin fur Computer Technik. — 1991.– № 12.
4. Д.А. Береза. Графические фонты фирмы Borland//Мир ПК.– 1992.– № 8.

Листинг 1. Структура префикса фонта

```
BgiFontPrefix = RECORD                { Префикс фонта }
    FontSignature      : ARRAY[1..4] of CHAR;
                  { Сигнатура фонта: PK#8#8 }
    FontDesription     : ARRAY[1..32] of CHAR;
                  { Описание фонта, поле заканчивается
                  символами #13#10 }
    Copyright          : ARRAY[1..32] of CHAR;
                  { Авторские права на фонт, поле
                  завершается символами #13#10 }
END;
PBgiFontPrefix = ^BgiFontPrefix;
                  { Указатель на префикс фонта } ■
```

Листинг 2. Структура заголовка BGI фонта

```
BgiFontHeader = RECORD          { Заголовок фонта }
    SigNature          : BYTE;  { Сигнатура заголовка:
                                  символ $1A, по нему
                                  ищем начало заголовка }
    HeaderSize         : WORD;  { Размер заголовка}
    FontName           : ARRAY[1..4] of CHAR;
                                  { Имя фонта }
    FontSize           : WORD;  { Размер фонта без
                                  учета заголовка }
    FontVersion        : BYTE;  { Версия фонта, поле
                                  не должно быть
                                  нулевым }
    FontModification   : BYTE;  { Модификация фонта }
    BgiInformation     : WORD;  { Информация
                                  о версии BGI }
END;
PBgiFontHeader = ^BgiFontHeader;
                  { Указатель на заголовок фонта } ■
```

Листинг 3. Структура описателя BGI фонта

```
BGIFontDsc = RECORD             { Описатель фонта }
    Signature          : CHAR;  { Сигнатура описателя —
                                  всегда символ "+" }
    SymbolsInFont      : WORD;  { Число символов
                                  в фонте }
    Stub01             : BYTE;  { Не используется,
                                  обычно 0 }
    FirstSymbol        : BYTE;  { ASCII код первого
                                  символа в фонте }
    StrokesOffset      : WORD;  { Смещение массива
                                  описаний символов
                                  фонта }
    FillFlag           : BYTE;  { Флажок заливки
                                  внутренней части
                                  символа, обычно равен 0 }
    CapitalHeight      : SHORTINT;  { Высота
                                  прописной буквы }
    BaseLine           : SHORTINT;  { Положение базовой
                                  линии, обычно 0 }
    Descender          : SHORTINT;  { Высота нижнего
                                  выноса }
    Stub02             : ARRAY[1..5] of BYTE;
                  { Не используются и заполнены 0 }
END;
PBgiFontDsc = ^ BgiFontDsc;                ■
```

Листинг 4. Таблицы BGI фонта

```
TYPE
    OffSetTable    = ARRAY[0..255] of WORD;
                   { Таблица смещений описаний символов
                   в фонте }
    POffSetTable   = ^OffSetTable;
    WidthTable     = ARRAY[0..255] of BYTE;
                   { Таблица ширины символов }
    PWidthTable    = ^WidthTable;
    Vector         = RECORD      { Вектор, используемый
                                   для отображения символов }
                     x,y : SHORTINT;
                     END;
    Vectors = ARRAY [0..MaxInt-1] of Vector;
                   { Фиктивный массив векторов }
PVector = ^Vectors;                        ■
```

Листинг 5. Базовые средства для работы с BGI фонтами

```
{*********************************************** }
{ List05.TXT — базовые средства для работы      }
{ с BGI фонтами                                 }
{ (BGI-Borland Graphics Interface)              }
{ (C) Тихонов А.И.,1992                         }
{*********************************************** }
UNIT List05;
INTERFACE
USES GRAPH;
CONST
    MaxSize = MaxInt * 2 + 1;
          { Максимальный размер массива }
TYPE
    DefaultArray   = ARRAY [0..MaxSize-1] of BYTE;
          { Фиктивный массив для вычисления адресов}
    PDefaultArray = ^DefaultArray;
          { Указатель на фиктивный массив }
    {$I LIST01.TXT}   { Описание префикса фонта }
    {$I LIST02.TXT}   { Описание заголовка фонта }
    {$I LIST03.TXT}   { Описатель фонта }
    {$I LIST04.TXT}   { Таблицы фонта }
TYPE
    BGIFont = OBJECT
        CONSTRUCTOR LoadBgiFont(FontName:STRING);
        DESTRUCTOR Done; VIRTUAL;
        FUNCTION SymbolHeight:INTEGER; VIRTUAL;
              { Высота заглавной буквы }
        FUNCTION SymbolFullHeight:INTEGER; VIRTUAL;
              {Полная высота с учетом нижнего выноса }
        FUNCTION SymbolWidth (C:CHAR):INTEGER;
              VIRTUAL;  { Ширина символа C в пикселах}
        FUNCTION StringLength(Gap:INTEGER;
                    S:STRING):INTEGER;VIRTUAL;
              { Длина строки в пикселах }
        PROCEDURE Transform(x1,y1,x,y:INTEGER;
              VAR XX,YY :INTEGER);VIRTUAL;
              { Преобразование координат }
```

Листинг 5 (продолжение)

```
        PROCEDURE OutSymbol(x1,y1:INTEGER;
                  Fill:Boolean;C:CHAR);VIRTUAL;
                  { Вывод символа C}
        PROCEDURE OutString(x0,y0,Gap:INTEGER;
                  Fill:BOOLEAN;S:STRING);VIRTUAL;
                  { Вывод строки }

    PRIVATE
        Loaded,
            { Флажок загрузки фонта }
        GotMemory    : BOOLEAN;
            { Флажок резервирования памяти под фонт }
        FontFile     : FILE;
            { Файл, в котором хранится фонт}
        FileLen      : LONGINT;
            { Размер файла с фонтом}
        FontPtr      : POINTER;
            { Указатель на начало описания фонта
            в оперативной памяти}
        PrefPtr      : PBgiFontPrefix;
            { Указатель префикса фонта }
        DefaultPtr   : PDefaultArray;
            { Указатель на фиктивный массив,
            используемый для вычисления адресов }
        BgiHeaderPtr : PBgiFontHeader;
            { Указатель на заголовок фонта }
        BgiFontDscPtr: PBGIFontDsc;
            { Указатель на начало описания символов }
        Offs         : POffSetTable;
            { Указатель на таблицу смещений описаний
              каждого из символов }
        Widths       : PWidthTable;
            { Указатель на таблицу ширины символов }
        Vect         : PVector;
            { Указатель на описание первого символа }

    PROCEDURE SetFontPtrs;
            { Базирование указателей на структуры фонта }
    FUNCTION CheckFont:BOOLEAN;
            { Проверка корректности фонта }
END;
PBGIFont = ^BGiFont;

IMPLEMENTATION

CONSTRUCTOR
BGIFont.LoadBgiFont(FontName:STRING);
            { Конструктор }

VAR
    p          : POINTER;
    MaxMem     : LONGINT;
    NumRead    : WORD;

BEGIN
    FontPtr := NIL;
    {$I-}
    ASSIGN(FontFile,FontName);
    RESET(FontFile,1);
    GotMemory := FALSE;
    IF (IoResult = 0) THEN BEGIN
        { Файл с именем FontName существует и открыт}
        FileLen := FILESIZE(FontFile);
        MaxMem:=MAXAVAIL;
        { Сколько памяти доступно? }
        IF FileLen < MaxMem THEN BEGIN
            GetMem(FontPtr,FileLen);
            GotMemory:=TRUE;
            BLOCKREAD(FontFile,FontPtr^,
                      FileLen,NumRead);
            IF NumRead = FileLen THEN BEGIN
                SetFontPtrs;      { Установка указателей }
                Loaded:= CheckFont;
            { Проверка корректности загруженного фонта }
            END
            ELSE BEGIN
                Loaded := FALSE;
                Done;
            END;
        END
        ELSE Loaded:=FALSE;
    END
    ELSE Loaded:=FALSE;
    {$I+}
END;

DESTRUCTOR BGIFont.Done;              { Деструктор }
BEGIN
    IF GotMemory THEN FREEMEM(FontPtr,FileLen);
    GotMemory := FALSE;
    Loaded := FALSE;
END;

PROCEDURE BGIFont.SetFontPtrs;
            { Базирование указателей на структуры фонта }
VAR
    I, J, K, L    : WORD;

BEGIN
    PrefPtr := FontPtr;
    DefaultPTR := FontPtr;

{
    Начало заголовка фонта не фиксировано, его
    поиск осуществляется по символу $1A
}
    I:=0;
    WHILE((DefaultPtr^[I]<>$1A) and (I<=300)) DO BEGIN
        Inc(I);
    END;

    BgiHeaderPtr := @DefaultPtr^[I];
            { Указатель на заголовок фонта }
    BgiFontDscPtr :=
    @DefaultPtr^[BgiHeaderPtr^.HeaderSize];
            { Указатель на описание фонта }
    Offs := @DefaultPtr^[BgiHeaderPtr^.HeaderSize +
            SIZEOF(BgiFontDscPtr^)];
        { Указатель на таблицу смещений описаний
        каждого из символов }
    Widths := @DefaultPtr^[BgiHeaderPtr^.HeaderSize +
            SIZEOF(BgiFontDscPtr^) +
            2*BgiFontDscPtr^.SymbolsInFont];
            { Указатель на таблицу числа векторов,
            используемых для описания каждого символа }
    Vect := @DefaultPtr^[BgiHeaderPtr^.HeaderSize +
            BgiFontDscPtr^.StrokesOffset];
            { Указатель на описание векторного
            представления первого символа }
END;

FUNCTION BGIFont.SymbolHeight:INTEGER;
            { Высота заглавной буквы }
BEGIN
    SymbolHeight:= BGIFontDscPtr^.CapitalHeight -
BGIFontDscPtr^.BaseLine;
END;

FUNCTION BGIFont.SymbolFullHeight:INTEGER;
            { Полная высота с учетом нижнего выноса }
```

```
BEGIN
    SymbolFullHeight:= BGIFontDscPtr^.CapitalHelght -
                       BGIFontDscPtr^.DesCender;
END;

FUNCTION BGIFont.SymbolWidth (C:CHAR):INTEGER;
          { Ширина символа C в пикселах}
BEGIN
    SymbolWidth:=
          Widths^[ORD(C)-BGIFontDscPtr^.FirstSymbol];
END;

FUNCTION BGIFont.CheckFont:BOOLEAN;
          { Проверка корректности фонта }
BEGIN
    CheckFont :=
        (PrefPtr^.FontSIgnature[1] = 'P')      AND
        (PrefPtr^.FontSignature[2] = 'K')      AND
        (PrefPtr^.FontSignature[3] = #8 )      AND
        (PrefPtr^.FontSignature[4] = #8 )      AND
        (BgiHeaderPtr^.Signature = $1A)        AND
        (BgiFontDscPtr^.Signature = '+');
END;

PROCEDURE BGIFont.Transform(x1,y1,x,y:INTEGER;
                VAR XX,YY :INTEGER);
          { Преобразование координат }
BEGIN
    XX := x1 + x;
    YY := y1 - y;          { Отсчет идет сверху }
END;
FUNCTION BGIFont.StringLength(Gap:INTEGER;
                S:STRING):INTEGER;
          { Длина строки в пикселах }
VAR
    I,L  : WORD;
BEGIN
    L:=0;
    FOR I := 1 TO LENGTH(S) DO
        L := L + SymbolWidth(S[I]) + Gap;
    StringLength:=L;
END;

PROCEDURE BGIFont.OutSymbol(x1,y1:INTEGER;
                Fill:Boolean;C:CHAR); { Вывод символа }
VAR
    cc          : char;
    I, J, K, JF  : WORD;
    xx,yy,ux,
    uy,mx,my   : INTEGER;
    PolyLine    : ARRAY[1..255] OF POINTTYPE;
          { Контур символа при его заливке }

BEGIN
    I:= ORD(c) - BGIFontDscPtr^.FIrstSymbol;
    IF I<= BGIFontDscPtr^.SymbolsInFont THEN
        K := I                 { Проверка номера символа }
    ELSE K := 0;

    J := Offs^[K] div 2;
          { Смещение описания символа C в словах }
    Self.Transform(x1,y1,0,0,mx,my);
    MOVETO(mx,my);
          { Начальное позиционирование символа }
    JF:= 1; { Начальное число точек в полилинии }
    PolyLine[JF].x:=mx;
    PolyLine[JF].y:=my;

    REPEAT
    WITH Vect^[J] DO BEGIN

        xx:= x AND $7F;

{ Горизонтальная координата текущего вектора}
        {$R-}
        IF (y AND $40)<>0 THEN
            { Вертикальная координата текущего вектора}
            yy:= y or $FF80
            { Преобразование отрицательного
            SHORTINT в INTEGER}
        ELSE yy:= y and $7F;
        { Преобразование положительного
        SHORTINT в INTEGER}
        {
            Преобразование координат
        }

        Transform(x1,y1,xx,yy,mx,my);

        IF (Fill AND (BgiFontDscPtr^.FillFlag<>0)) THEN
            { Заполнение символа }
        BEGIN
            IF (((y AND $80) = 0) OR
                ((Vect^[j].x AND $80) = 0)) THEN
            BEGIN                          { Закраска }
                IF JF>2 THEN FILLPOLY(JF,PolyLine);
            { Закраска только тогда, когда в контуре
            больше двух точек }
                JF :=0;
                IF (y AND $80) = 0 THEN BEGIN
                { Перо поднято }
                    JF := 1;
                    PolyLine[JF].x:=mx;
                    PolyLine[JF].y:=my;
                    MOVETO(mx,my);
                END;
            END
            ELSE BEGIN
            { Добавление вектора в контур символа }
                INC(JF);
                PolyLine[JF].x:=mx;
                PolyLine[JF].y:=my;
                LINETO(mx,my);
            END
        END
        ELSE BEGIN                 { Вывод символа }
            IF (y AND $80) = 0 THEN
                MOVETO(mx,my)   { Сдвиг к точке mx,my }
            ELSE
                LINETO(mx,my);    { Вывод вектора }
        END;

        INC(J);
            { Переход к следующему вектору}
    END;
    UNTIL ((Vect^[j].x and $80) = 0);
END;

PROCEDURE BGIFont.OutString(x0,y0,Gap:INTEGER;
          Fill:BOOLEAN;S:STRING);     { Вывод строки }
VAR
    I,L,X1,Y1   : INTEGER;

BEGIN
    X1:=X0;
    Y1:=Y0;
    FOR I:=1 TO LENGTH(S) DO BEGIN
        OutSymbol(x1,y1,Fill,S[I]);
        x1 :=x1 + SymbolWidth(S[I]) + Gap;
    END;
END;

END.
```

Листинг 6. Объект Font1

```
UNIT LIST06;

INTERFACE

USES GRAPH, LIST05;

TYPE

{$IFOPT N+}
    FLOAT = DOUBLE;
{$ELSE}
    FLOAT = REAL;
{$ENDIF}

FONT1 = OBJECT(BgiFont)
{ Допускается изменение масштаба и поворот надписи }
    MX     : FLOAT;   { Масштаб по X }
    MY     : FLOAT;   { Масштаб по Y }
    AIFA   : FLOAT;   { Угол поворота строки в градусах}
    X0,Y0  : FLOAT;   { Координаты центра вращения }
    SN,CN : FLOAT;   { Синус и косинус угла поворота }

PROCEDURE SetParameters1(MXX,MYY,A:FLOAT;
        X00,Y00:INTEGER); VIRTUAL;
            { Инициализация параметров
            линейного преобразования }
FUNCTION SymbolHeight:INTEGER; VIRTUAL;
            { Высота заглавной буквы }
FUNCTION SymbolFullHeight:INTEGER; VIRTUAL;
            { Полная высота с учетом нижнего выноса }
FUNCTION SymbolWidth (C:CHAR):INTEGER;VIRTUAL;
            { Ширина символа C в пикселах}
FUNCTION StringLength(Gap:INTEGER;
    S:STRING):INTEGER;VIRTUAL;
            { Длина строки в пикселах }
PROCEDURE Transform(x1,y1,x,y:INTEGER;
    VAR XX,YY :INTEGER);VIRTUAL;
            { Преобразование координат }
PROCEDURE OutString(xx0,yy0,Gap:INTEGER;
    Fill:BOOLEAN;S:STRING);VIRTUAL;  { Вывод строки }
END;

IMPLEMENTATION

PROCEDURE
Font1.SetParameters1(MXX,MYY,A:FLOAT;
        X00,Y00:INTEGER);
        { Инициализация параметров преобразований }

BEGIN
    MX := MXX;
    MY := MYY;

    SN :=SIN(PI*A/180.0);
    CN :=COS(PI*A/180.0);
    X0 :=X00;
    Y0 :=Y00;
END;

FUNCTION Font1.SymbolHeight:INTEGER;
            { Высота заглавной буквы }
BEGIN
    SymbolHeight := ROUND(BgiFont.SymbolHeight * MY);
END;

FUNCTION Font1.SymbolFullHeight:INTEGER;
            { Полная высота с учетом нижнего выноса }
BEGIN
    SymbolFullHeight := ROUND(BgiFont.SymbolHeight *
                                        MY);
END;

FUNCTION Font1.SymbolWidth (C:CHAR):INTEGER;
            { Ширина символа C в пикселах}
BEGIN
    SymbolWidth := ROUND((BgiFont.SymbolWidth(C)) * MX);
END;

FUNCTION Font1.StringLength(Gap:INTEGER;
    S:STRING):INTEGER;      { Длина строки в пикселах }
BEGIN
    StringLength:= BGIFont.StringLength(ROUND(GAP *
                                        MX),S);
END;

PROCEDURE Font1.Transform(x1,y1,x,y:INTEGER;
    VAR XX,YY :INTEGER);
            { Преобразование координат }
VAR
    X2,Y2  : FLOAT;

BEGIN

    X2 := x1 + x * MX - x0;
    Y2 := y1 - y * MY - y0;
    XX := ROUND(x0 + CN * X2 + SN * Y2);
    YY := ROUND(y0 - SN * X2 + CN * Y2);
END;

PROCEDURE Font1.OutString(xx0,yy0,Gap:INTEGER;
    Fill:BOOLEAN;S:STRING);              { Вывод строки }
BEGIN
    BgiFont.OutString(xx0,yy0, ROUND(Gap * MX), Fill, S);
END;

END.
```

Листинг 7. Объект Font2

```
UNIT LIST07;

INTERFACE

USES LIST06;

TYPE
{$IFOPT N+}
    FLOAT = DOUBLE;
{$ELSE}
    FLOAT = REAL;
{$ENDIF}

    FONT2 = OBJECT(FONT1)
        { Надпись по дуге окружности }
        R       : FLOAT;   { Радиус дуги }
        XC,YC : FLOAT;   { Координаты центра дуги }
        DY      : FLOAT;
        L2      : FLOAT;   { Половина длины строки }
        DSIGN,
        H   : FLOAT;

PROCEDURE
SetParameters2(MXX,MYY:FLOAT;DYY:INTEGER;
            X00, Y00:INTEGER); VIRTUAL;
                        { Инициализация параметров }
                        { DYY - максимальная высота дуги }
PROCEDURE Transform(x1,y1,x,y:INTEGER;
        VAR XX,YY :INTEGER);VIRTUAL;
                        { Преобразование координат }
PROCEDURE OutString(xx0,yy0,Gap:INTEGER;
                            Fill:BOOLEAN;S:STRING);
                            VIRTUAL;       { Вывод строки }
END;
```

```
IMPLEMENTATION

PROCEDURE
Font2.SetParameters2(MXX,MYY:FLOAT;DYY:INTEGER;
          X00,Y00:INTEGER);
       { Инициализация параметров преобразований }

BEGIN
    Font1.SetParameters1(MXX,MYY,0.0,X00,Y00);
    DY := DYY;
END;

PROCEDURE Font2.Transform(x1,y1,x,y:INTEGER;
    VAR XX,YY :INTEGER);
          { Преобразование координат }
VAR
    XA,YA          : INTEGER;
    HH,XS,YS,RS  : FLOAT;
BEGIN
    Font1.Transform(x1,y1,x,y,XA,YA);
    XS := XA - xc;
    RS := R + DSIGN * (YA-y0);
    HH := SQRT(H * H + XS * XS);
    XX := ROUND(xc + XS * RS / HH);
    YY := ROUND(yc + H * RS / HH * DSIGN);
END;

PROCEDURE Font2.OutString(xx0,yy0,Gap:INTEGER;
       Fill:BOOLEAN;S:STRING);     { Вывод строки }

VAR
    LL : WORD;

BEGIN
    LL := Font1.StringLength(Gap,S);
    L2 := LL /2.0;
    IF DY<>0 THEN BEGIN
       IF LL<>0 THEN BEGIN
       { Длина строки не равна нулю }
          IF DY>0 THEN DSIGN:=1.0 ELSE DSIGN:=-1.0;
          R := (L2 * L2 + DY * DY)/ 2.0 / ABS(DY);
          XC := xx0 + L2;
          YC := yy0 - DSIGN * R + DY;
          H := R - ABS(DY);
          Font1.OutString(xx0,yy0,Gap,Fill,S);
       END;
    END;
END;

END.
```

Листинг 8. Тестовая программа

```
PROGRAM LIST08;
{$N+,E+}
{
Пример использования средств работы с фонтами BGI
}

USES
    CRT, GRAPH, LIST05, LIST06, LIST07;

CONST

    Script=' ... и надпись написал... ';

VAR
    Gd,Gm : INTEGER;
    C       : CHAR;
    F       : BGIFont;
    F1      : Font1;
    F2      : Font2;
    H,HF,
    W,WW : INTEGER;
    I       : INTEGER;

BEGIN

    {
       Параметры фонта BOLD
    }
    F.LoadBgiFont('BOLD.CHR');
    H := F.SymbolHeight;
    HF := F.SymbolFullHeight;
    W := F.SymbolWidth('O');
    WW := F.StringLength(1,Script);
    ClrScr;

    GOTOXY(20,25); WRITE
    ('Для продолжения нажмите любую клавишу...');

    GOTOXY(17,10); WRITE
    ('Тестирование средств для работы с BGI фонтами');
    GOTOXY(20,12); WRITE('Фонт: BOLD.CHR');
    GOTOXY(20,13); WRITE('Высота заглавных букв:',H:4);

    GOTOXY(20,14); WRITE
    ('Полная высота символов:',HF:4);
    GOTOXY(20,15); WRITE('Ширина символа O:',W:4);

    GOTOXY(20,16); WRITE
    ('Длина строки "',Script,'":',WW:4);

    C:=ReadKey;

    Gd := Detect; InitGraph(Gd, Gm, '');
    IF GraphResult <> grOk THEN Halt(1);

    {
       Вывод рис.3
    }
    F.OutString(10, 200,5,FALSE,Script);
    F.Done;
    C:=ReadKey;
    ClearDevice;

    {
       Вывод рис.4
    }
    F1.LoadBgiFont('EURO.CHR');
    FOR I:=0 TO 12 DO BEGIN
       F1.SetParameters1(0.20,0.15,30.0*I,320,170);
       F1.OutString(320,170,2,FALSE,Script);
    END;
    F1.Done;
    C := READKEY;
    CLEARDEVICE;

    {
       Вывод рис.5
    }
    F2.LoadBgiFont('BOLD.CHR');
    F2.SetParameters2(1.0,1.0,100,0,200);
    F2.OutString(0,200,5,FALSE,Script);
    F2.SetParameters2(1.0,1.0,-100,0,200);
    F2.OutString(0,200,5,FALSE,Script);
    F2.Done;

    C:=ReadKey;

    CloseGraph;
END.
```

Немного о дисассемблерах. (Опыт работы с программой SOURCER)

Л. Ливеровский

Видимо, не надо объяснять читателю, что в практической работе программиста, особенно отечественного, время от времени возникает необходимость восстановить исходный текст программы по загрузочному модулю.

Что значит восстановить исходный текст? Мы будем здесь рассматривать только самый простой случай.

Самым простым является восстановление исходного текста программы на язык самого низкого уровня, имеющегося на данной машине. Таким языком является язык ассемблера, а программы, восстанавливающие исходный текст, называются дисассемблерами.

Однако в общем виде эта задача для традиционных ЭВМ не может быть решена с помощью другой программы, ничего не знающей о том, что и как делает программа, текст которой надо восстановить. С чем это связано?

Во-первых, в процессе трансляции программы уменьшается избыточность информации, содержащейся в ней. Теряются комментарии, содержательные для программиста символические имена. Более того, исчезает разница между константами и перемещаемыми выражениями (метками команд и данных, которые представляют собой относительные адреса в исходной программе и оказываются представленными константами в исполняемом модуле).

Во-вторых, большинство современных ЭВМ (и IBM PC не является исключением) есть компьютеры с архитектурой фон Неймана, или ЭВМ с хранимой программой. Основной отличительной чертой таких ЭВМ является хранение как команд центрального процессора, так и данных, которыми эти команды оперируют, в общей памяти с единой адресацией, называемой часто оперативной памятью.

При такой организации ЭВМ нельзя сказать, что содержит ячейка памяти (команду или данные), до тех пор пока к ее содержимому не обратится центральный процессор за командой или за операндом команды. Более того, в разные моменты времени выполнения одной программы центральный процессор может обращаться к одной и той же ячейке памяти то за командой, то за данными.

Такое построение ЭВМ делает ее чрезвычайно гибким инструментом, позволяющим одной программе создавать другие программы, писать размножающиеся и саморазвивающиеся программы (вирусы), делать самомодифицирующиеся программы (самораспаковывающиеся выполняемые программы).

Вместе с тем это не дает возможности однозначно восстановить исходный текст даже на языке ассемблера, поскольку для восстановления исходного текста нужно как бы понять, что делает программа. А сделать это, не имея какой-то априорной информации о программе, возможно, только смоделировав ее поведение во всех возможных случаях, число которых может быть

столь огромно, что не уложится не только в головы программистов, но и в "железные мозги" самых современных супер-ЭВМ.

Но даже если бы и удалось смоделировать все возможные варианты поведения программы, то осталась бы не менее сложная задача выявления того, какие варианты являются "осмысленными", а какие определяются невозможными (или редко встречающимися) последовательностями входных данных либо просто ошибками и неточностями в программе.

Однако теория и практика программирования показали, что для решения очень широкого класса задач смешение кода и данных является не только ненужным, но часто даже и вредным делом. Поэтому был выработан принцип программирования, в котором четко различаются не только команды и данные вообще, но и разные виды данных между собой (типизация данных).

С использованием этого принципа были построены многие языки программирования высокого уровня (Algol, Pascal, Modula-2) с сильной типизацией данных.

Существуют операционные системы, которые всегда "знают", где у программы код, а где данные. Это очень важно для многозадачных систем: одна копия кода в памяти в таком случае может выполняться несколькими процессами одновременно.

С учетом сказанного наша задача уже не кажется такой безнадежной.

Поэтому, располагая некоторой информацией о программе и средствах, с помощью которых она была сделана, необходимо иметь дисассемблер, которому эту информацию можно сообщить.

Одним из лучших таких дисассемблеров для IBM PC, получившим распространение у нас в стране, является программа SOURCER (внешний вид экрана представлен на рис. 1, 2).

Как работает SOURCER?

Основная задача, которую пытается решить SOURCER, — разделить программу на области кода и данных. Для этого он использует как некие предположения, основанные на типе дисассемблируемой программы, так и явные указания пользователя. Эти явные указания в числе прочих в виде команд помещаются в файл определений или def файл.

SOURCER может автоматически снабжать комментариями адреса портов ввода-вывода, зарезервированные области памяти BIOS, программные прерывания DOS (включая многие недокументированные), BIOS, EMS, DPMI, VCPI.

Все эти комментарии занимают оперативную память во время работы SOURCER.

Пользователь может, задавая команды в файле определения комментариев к оборудованию (HARDWARE.TBL), удалять определенные комментарии и их группы, добавлять свои комментарии к портам ввода-вывода, программным прерываниям, областям памяти. Смотри описание формата HARDWARE.TBL в приложении.

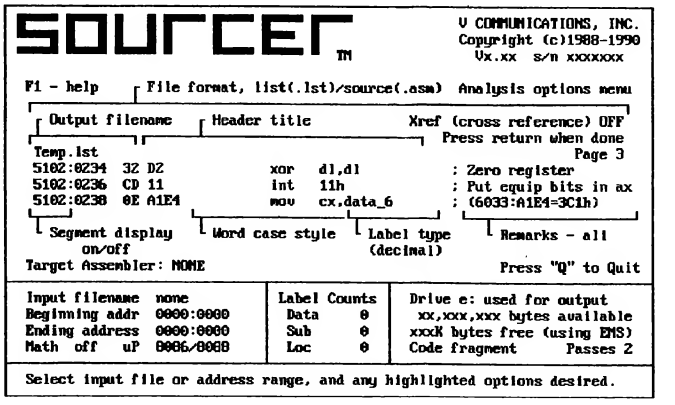


Рис. 1. Диалоговый экран задания команд

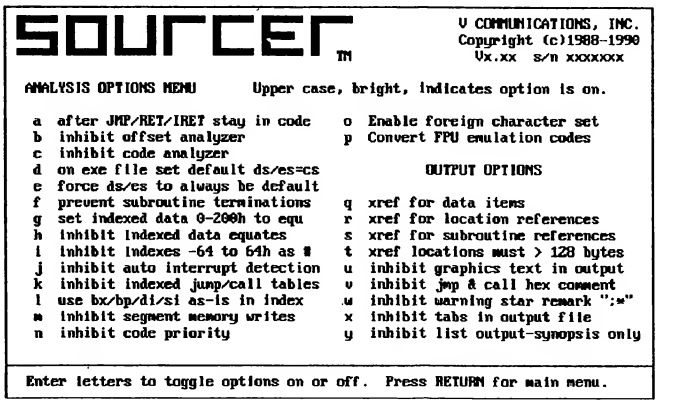


Рис. 2. Диалоговый экран задания флагов дисассемблирования

Общая схема информационных связей приведена на рис. 3.

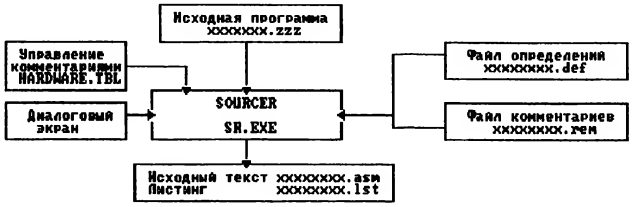


Рис. 3.

Когда SOURCER считает, что некоторая область содержит код/данные, мы будем говорить, что он находится в режиме кода/данных, соответственно.

После загрузки исходного файла в память SOURCER пытается определить количество сегментов в программе, их адреса и длину.

Для этого ему необходимо определить точки входа в программу и содержимое сегментных регистров (если в них есть осмысленная информация) в момент передачи управления программе. При этом SOURCER руководствуется параметром CODE STYLE (тип кода), который определяет, как найти точку входа в программу, и начальные значения регистров при запуске программы. SOURCER различает следующие типы кода и соответствующие им типы программ.

COM — программа DOS формата COM

Точка входа находится по смещению 100h от начала кодового сегмента. Имеет один сегмент и не требует настройки адресов при загрузке (так называемый позиционно независимый формат). Однако большинство программ формата COM обращаются к памяти за пределами своего сегмента, и SOURCER разрешает ссылки на такие данные правильно, создавая исходный текст односегментной программы, удовлетворяющий требованиям формата COM файлов. После загрузки в память программами DOS регистры получают следующие значения: CS,DS,SS,ES — сегмент PSP, SP — конец сегмента PSP (если достаточно памяти, то FFFEh), IP=100h.

EXE — программа DOS формата EXE

Точка входа определяется из заголовка EXE файла. Требует настройки адресов после загрузки в память (segment fixup). Может иметь произвольное количество сегментов (определяется возможностями компоновщика). SOURCER поддерживает дисассемблирование программ, содержащих не более 250 сегментов. После загрузки в память программы регистры получают следующие значения: DS, ES — сегмент PSP, CS:IP — адрес точки входа, SS:SP — адрес стека из заголовка EXE файла.

DEVICE DRIVER — загружаемый драйвер DOS

Загружается по команде DEVICE файла CONFIG.SYS. Имеет две точки входа, определяемые по заголовку драйвера. Файлы, имеющие расширения COM, BIN, SYS, проверяются SOURCER на наличие заголовка драйвера. При вызове драйвера по любой из точек входа пара CS:IP содержит адрес точки входа, который берется из заголовка драйвера, SS:SP содержит адрес стека DOS.

ZERO START (OVERLAY)

Внутренние программы DOS или фрагменты программ (оверлеи и тому подобные), аналогичные по структуре программам формата COM, но имеющие точку входа со смещением 0 от начала.

FRAGMENT

SOURCER не пытается определить точки входа в программу и границы сегментов, а считает все кодом, идущим прямо от начала файла. В качестве данных дисассемблируются те участки программы, на которые есть явные ссылки как на данные.

CODE STYLE задается с помощью команды или берется из файла определений. Точки входа в программу можно сообщить SOURCER дополнительно, используя директивы файла определений (def файл).

После определения точек входа в программу SOURCER пытается определить количество сегментов в программе и их границы. Каждому определенному сегменту соответствует сегментная псевдопеременная SOURCER, содержащая адрес сегмента. Сегментные переменные могут использоваться в def файле везде, где необходимо указание адресов сегментов. Это позволяет сделать def файл независимым от фактического места загрузки программы в память. Сегментные переменные имеют имена SEG_A ... SEG_Z, SEG_AA ... SEG_AZ ... SEG_IO. Сегментная переменная SEG_A определяет сегмент с наименьшим адресом, seg_B с наименьшим после seg_A и так далее.

Дисассемблирование проводится за несколько проходов (от 2 до 9). Во время каждого прохода строится таблица символов с указанием, какие символы представляют собой ссылки на данные (переменные), какие — ссылки на команды (метки), какие — ссылки на подпрограммы (процедуры). Таблица символов сортируется между проходами, удаляются неверно определенные ссылки. Каждый проход анализирует ссылки на данные и код, пытаясь установить границы областей кода и данных.

Последний проход формирует необходимые директивы ассемблера, в соответствии с определенными сегментами и таблицей символов, форматирует строки листинга и присоединяет к ним комментарии, как внутренние (вызовы DOS, BIOS, адреса портов B/B), так и внешние и выводит листинг в файл. Внешние комментарии могут определяться пользователем в специальном файле комментариев или rem файле.

Во время последнего прохода работа анализатора кода и данных продолжается. В некоторых случаях на последнем проходе выясняется, что ссылка на данные должна быть ссылкой на команду или наоборот. Если при этом соответствующая строка листинга уже была выведена на диск, то при повторном ассемблировании возникнет ошибка.

Большее количество проходов должно улучшать качество дисассемблирования. Это безусловно относится к программам, полученным компиляцией с традиционных языков программирования (C, Pascal).

Объясняется это тем, что компиляторы создают отдельные сегменты для разных типов данных (глобальные, глобальные инициализированные, статические и так далее) и для кода. Код и данные в таких программах не перемешаны. Трудности могут встретиться во время дисассемблирования подпрограмм библиотек run-time. Такие подпрограммы часто кодируются вручную на ассемблере и содержат в себе всякого рода "трюки".

А вот при дисассемблировании программ, содержащих участки самомодифицирующегося кода, количество проходов больше двух смысла не имеет. По умолчанию для типа кода FRAGMENT SOURCER выбирает значение количества проходов (PASSES), равное 2.

Если попробовать дисассемблировать SOURCER EXE программу, "свернутую" архиватором

PKLITE.EXE, то ничего не выйдет. С такого рода программами SOURCER не справляется, и понятно почему.

В SOURCER имеется подсистема, моделирующая работу программы. Она отслеживает содержимое регистров и ведет отдельный стек для программы.

Моделирование работы программы проводится последовательно. Игнорируются все переходы, циклы и записи в память. Команды чтения из памяти обрабатываются. Моделирующая программа пытается определить, какое значение взято из памяти. (Но как это наверняка сделать, если не знать, что в память было записано?)

Когда встречается вызов подпрограммы, регистры модели запоминаются и код подпрограммы обрабатывается с сохраненными значениями регистров. По окончании обработки подпрограммы предыдущие значения регистров модели восстанавливаются.

Создаваемые SOURCER метки данных (ссылки на данные — data references) получают имена вида data_xxxx, метки команд (ссылки на команды — location references): loc_xxxx, метки процедур (ссылки на подпрограммы — subroutine references): sub_xxxx.

Применяемое в SOURCER моделирование работы программы в состоянии правильно распознавать индексированные переходы и вызовы подпрограмм, такие как call [bx+480].

Когда имитатору работы программы встречается подобного вида инструкция, содержимое регистров модели используется в попытке определить начало таблицы вызовов (переходов). Для определения длины таблицы используется собственный метод, требующий для своей работы не менее трех проходов по программе. Имитатор пытается воссоздать код вида:

```
                call  TABLE_OF_CALLS[bx]
          ...
TABLE_OF_CALLS label word
                dw    PROC1
                dw    PROC2

                dw    PROCn
```

Он "понимает" четыре типа таблиц (см. описание def файла в приложении).

Работой SOURCER могут эффективно управлять команды, помещаемые в файл определений (def файл). Все команды разделяются на следующие три группы:

- управление режимами дисассемблирования;
- определение областей;
- определение ссылок;

Восстановление исходного текста программы с помощью SOURCER проводится в несколько этапов. На каждом этапе программист анализирует полученный листинг, пытается понять, что делает программа, и корректирует def файл. Первая часть работы заканчивается, когда после ассемблирования и компоновки программы получается код, идентичный первоначальному.

Как правило, работа на этом не заканчивается. Часто бывает необходимо внести исправления в полученный текст. Тут надо быть очень осторожным. Возможно, в полученном тексте есть много мест, где символическое имя представляет собой константу, а должно быть перемещаемое выражение (ссылка на данные или команды относительно какого-либо сегмента) или наоборот. Или просто стоит числовая константа там, где необходимо перемещаемое выражение.

Внося изменения в полученный текст, следует руководствоваться следующими правилами.

Удаляя команду, заменять ее на необходимое количество инструкций nop.

Добавлять команды и данные надо в конец программы, чтобы не "потревожить" ссылки.

Добавляя команды или данные в произвольном месте, необходимо предварительно все ссылки на данные и команды, сделанные через константы, заменить перемещаемыми выражениями (метками команд и данных) относительно необходимых сегментов. Это может быть очень трудоемко или даже практически невозможно.

Сводка основных команд SOURCER приведена в приложении. ■

Монитор–Аспект — 1993

Монитор Аспект

Адрес _____

Ф.И.О. или название организации _____

Стоимость подписки на 1 номер журнала "Монитор–Аспект" — 250 рублей. В 1993 году будет выпущено 3 номера (годовая подписка — 750 руб.). Отметьте крестиками те номера журнала, которые Вы желаете получить. Перечислите сумму оплаты на счет НТО "Софт-Москва": р/с 467002 в КБ "Банк развития XXI век", к/с 161816 в РКЦ ГУ ЦБ РФ по г. Москве, МФО 201791. Вышлите подписной купон и квитанцию об оплате по адресу: 117419 Москва, а/я 765.

Приложение

Формат DEF файла SOURCER. Основные команды

Для запуска SOURCER с файлом определений в командной строке вместо имени дисассемблируемой программы указывается имя файла определений.
Все команды в def файле начинаются с первой позиции. Если в первой позиции стоит пробел, то такая строка считается строкой комментария.
Команды можно сократить до одной буквы.
Команды из каждой группы помещаются в свой раздел def файла. Признаком окончания раздела служит хотя бы одна строка комментария. Если в разделе нет никаких команд, то в нем помещается с первой позиции слово "none".
В описании def файла все числа шестнадцатиричные.

Раздел 1. Управление режимами

Analysis flag = xxxxx

Любые флаги. Символ верхнего регистра включает режим, нижнего выключает.

Подгруппа 1

Управляет определением областей кода и данных.
Когда какая-либо область становится недоступна как код (в нее нет передач управления), SOURCER переходит в режим данных. С помощью флагов этой группы его действия можно изменить.

A — после команд jump/ret/iret оставаться в "режиме кода".
B — выключить анализатор смещений.
C — отключить анализатор кода.

Подгруппа 2

Управляет содержимым регистров имитатора работы программы.
По умолчанию регистры DS и ES имитатора устанавливаются так же, как их устанавливает DOS после загрузки программы. С помощью флагов этой группы режим умолчания можно изменить.

D — в регистрах DS и ES имитатора установить значение кодового сегмента. DS,ES=CS.
E — игнорировать изменения содержимого регистров ES и DS. Считать, что регистры DS и ES имитатора содержат предварительно установленные значения (указания def файла перекрывают действие этого флага).

Подгруппа 3

Управляет дисассемблированием индексированных ссылок.
В некоторых случаях задание операнда команды в виде [si+345] может привести к тому, что SOURCER создаст ссылку на данные, которая может в некоторых случаях нарушить дальнейшее дисассемблирование. Будет создана ссылка на данные data_xxxx, связанная с сегментом данных: ds:345.

G — дисассемблировать индексные ссылки на данные в диапазоне 0-200h как константы (используя директиву ассемблера EQU для присвоения значения символу (идентификатору)).
H — запретить дисассемблировать индексные ссылки на данные как константы (используя директиву ассемблера EQU для присвоения).
I — запретить дисассемблирование индексных ссылок на код в диапазоне -64—64 как констант.

Приложение

Подгруппа 4

Управляет дисассемблированием.

F — запретить выявление подпрограмм, которые не возвращают управление в место вызова.
J — запретить автоматическое определение точек входа по прерываниям.
K — запретить анализ индексированных переходов и вызовов подпрограмм.
L — при анализе индексированных переходов и вызовов подпрограмм использовать любые значения индексных регистров имитатора (bx, bp, si, di).
M — запретить моделировать запись в память сегментных адресов.
N — выключить приоритет кода.
P — преобразовывать прерывания эмуляции сопроцессора в команды сопроцессора.

Подгруппа 5

Прочие флаги управления выводом.

O — разрешить расширенный набор символов в листинге (коды от 128).
Q — создавать перекрестные ссылки для всех ссылок на данные.
R — создавать перекрестные ссылки для всех ссылок на команды.
S — создавать перекрестные ссылки для всех ссылок на подпрограммы.
T — игнорировать перекрестные ссылки в диапазоне менее -128—128.
U — запретить графические символы в листинге.
V — запретить указывать в комментариях к командам call и jump абсолютный адрес ссылки.
W — запретить комментарий-предупреждение ;*.
X — запретить табуляции в листинге.

Code style = xxx

com, exe, fragment, zero start, device driver или overlay.

Drive = x

Диск для выдачи листинга.

Format = xxx

Выходной формат: LST — листинг (умолчание) или ASM — ассемблерный текст.

Go

Запуск программы в работу без вывода диалогового экрана.

Header = xxxxxx xxxx xxxx

Строка заголовка листинга (до 32 символов)

Input filename = xxxx

Имя входного файла. Может включать путь.

Keep segments = xxx

FILE — использовать сегменты, определенные после загрузки файла.
DEF — использовать сегменты, определенные в def файле.
BOTH — использовать сегменты, определенные и тут и там. Необходимо задавать эту команду до загрузки файла.

Приложение

Label = xxx

Выбор вида меток в листинге.

Decimal (десятичная)	data_6.
Zero fill (с ведущими нулями)	data_0006.
Segment & offset (сегмент:смещение)	d_6033_A1E4.
Letter segment & offset (буква от имени сегмента: смещение)	data_b_A1E4.

Lower case off

Все символы в верхнем регистре.

Math on

Дисассемблировать команды математического сопроцессора.

None

Секция не содержит команд.

Output filename = xxxx

Имя выходного файла. Может включать путь.

Passes = x

Количество проходов (от 2 до 9).

Remark = xxx

Виды встроенных комментариев к тексту:

All	—	все.
None	—	никакие.
Data remarks only	—	комментарии к данным (data references).
Interrupts & I/O only	—	комментарии к прерываниям (BIOS, DOS) и портам ввода-вывода.
Others only	—	прочие комментарии.
Except Data	—	все, кроме данных.
Except Interrupts	—	все, кроме прерываний.
Except Others	—	все, кроме прочих.

Segment display off

Выключить печать содержимого сегментных регистров в листинге.

Target assembler = xxx

Формировать текст для ассемблера:
MASM-4.0, MASM-5.0, MASM-5.1, TASM-1.0, TASM-2.0, OPTASM, OTHER или NONE.

Tabs off

Использовать в листинге пробелы вместо табуляций.

uP = xxxxx

Набор команд:
8088, 80186, 80286 (режим 8086), P80286 (инструкции защищенного режима), V20/V30, 80386,P386 (инструкции защищенного режима), 80486, P486 (инструкции защищенного режима).
AUTO — автоматическое определение типа микропроцессора (по умолчанию).

Vertical lines/page = xxx

Количество строк на странице листинга.

Приложение

Xref = xx

Перекрестные ссылки:
ON —включить (устанавливает 5 проходов),
OFF —выключить.

Раздел 2. Определение областей

Формат раздела:

начало	конец	умолчание		тип	размер	
seg:off	off	ds,	es	type	seg size	комментарий

segment:offset = начальный адрес (hex).
end offset = конечный адрес (смещение hex).
default ds, es = значение регистров при обращении к сегменту по умолчанию (если не будет перекрыто явными заданиями значений в программе).
segment type = auto — смешанный код/данные, автоматическое определение ссылок.
code — код, ограниченные функции определения ссылок на данные.
data — только данные.
stack — стек.
rauto — смешанный код/данные в ROM.
rcode — код в ROM.
segment size = use16 16-разрядные (умолчание, 8086 совместимые).
use32 32-разрядные.

Например:

начало seg:off	конец off	умолчание ds	es	тип type	размер seg size	комментарий
seg_a:0000	0001	seg_a	seg_a	code	use16	начало программы
seg_b:0000	0200	0000	0000	stack	use16	стек программы
0000:0000	0FFF	0000	0000	data	use16	таблица векторов прерываний

Раздел 3. Определение ссылок

Формат раздела

начало	тип	атрибуты и символическое имя
seg:off	type	options & label field

segment:offset = место расположения.
type = тип поля (области памяти)
Данные:
DB = байт данных.
DW = слово данных.
DD = двойное слово данных.

Приложение

DA = текстовая строка ascii.
DS = структура данных.

Команды:
SUB = подпрограмма.
LOC = метка команды (ссылка на команду).

Принудительный тип:
FORCE= принудительное задание атрибутов.

Options:

Для данных:
- C xxx — номер комментария (от 1 до 3FF).
- DUP — дублировать повторяющиеся байты.
- INDEX — заменяет индексную ссылку на символическое имя константы.
- MULTI — многократно определенная ссылка.

а) адрес подпрограммы:
- OSN — offset sub_xxx.
- OSF — offset sub_xxx, seg sub_xxx.

б) адрес команды:
- OLN — offset loc_xxx.
- OLF — offset loc_xxx, seg loc_xxx.

в) адрес данных
- ODN — offset data_xxx.
- ODF — offset data_xx, seg data_xx.
- R xxxx — повторить 0—FFFF раз.
- SEG — для dw только. Слово содержит имя (адрес) сегмента.
- UNUSED — если на область никогда не ссылаются, не включать метку в листинг.

Для команд:
- C xxx — номер комментария (от 1 до 3FF).
- EXT — точка входа в программу.
- FAR — процедура с атрибутом far.
- NEAR — процедура с атрибутом near.
- UNUSED — если на область никогда не ссылаются, не включать метку в листинг.

Для процедур:
- C xxx — номер комментария (от 1 до 3FF).
- FAR — процедура определена с атрибутом FAR.
- TERM — подпрограмма не передает управление в место вызова.
- UNUSED — если на область никогда не ссылаются, не включать метку в листинг.

Принудительный тип:
- CODE — переключиться в режим кода.
- DATA — переключиться в режим данных.
- COMMENT xxxxxxxxx— в листинге по этому адресу поставить комментарий.
- CODE16 — переключиться в 16-разрядный кодовый режим.
- CODE32 — переключиться в 32-разрядный кодовый режим.
- ENDP — поставить ENDP после этого адреса.
- <per>=xxxx — изменить значение любого регистра имитатора.
  <per>=ax/bx/cx/dx/bp/si/di/sp/ds/es/ss
- E<per>=xxxx — изменить значение любого 32-разрядного регистра имитатора.
  <per>=eax/ebx/ecx/edx/ebp/esi/edi
- EAx — тип ссылки для команды LEA:
  - EAS — ссылка на подпрограмму;
  - EAL — ссылка на команду;
  - EAD — ссылка на данные;

  сегмент также может быть задан:
  EAL seg_b
- Ox — преобразование непосредственного значения в смещение:
  - OS — для ссылки на подпрограмму;
  - OL — для ссылки на команду;
  - OD — для ссылки на данные;

  сегмент также может быть задан:
  OD seg_b
- REG — распечатать содержимое регистров имитатора.

Задание типа индексной таблицы переходов и вызовов:
- SN — таблица содержит адреса подпрограмм: "offset sub_xx";
- SF — таблица содержит адреса подпрограмм: "offset sub_xxx, seg sub_xxx";
- LN — таблица содержит адреса переходов: "offset loc_xx";
- LF — таблица содержит адреса переходов: "offset loc_xxx, seg loc_xxx";

Формат: "seg_a:xxxx f, sn 1234:22 34",
где
- 1234:22 — адрес таблицы из 34 элементов;
- label field = допустимое символическое имя для ассемблера до 15 символов.

Примеры команд DEF файла

seg:off	type	options & label		
seg_a:0028	loc,	C 4	begin_program	Начало программы, комментарий 4
seg_a:0043	loc		exit	
seg_c:0123	dw,	Index	Ind_123	Ссылка вида [si+123], стоящая по этому адресу, дисассемблируется как Ind_123[si]. Ind_123 определяется так: Ind_123 EQU 123
seg_c:0456	dw,	OSF	pointer_to_sub	Слово, содержащее адрес (смещение:сегмент) подпрограммы.
0000:0449	db	Video_Mode		Первичное определение.
0040:0049	db,	Multi	Video_Mode	Повторное определение. Не порождать метки данных или константы.
seg_a:0789	loc,	Ext	Keyboard_Int_Handler	Внешняя точка входа в программу.
seg_a:0345	sub,	Terminate	Error_exit	Подпрограмма, не возвращающая управление в точку вызова. После инструкции вызова SOURCER переключается в режим данных.
seg_a:0321	f,	ds=seg_c		Установка значений регистров имитатора. В команде по адресу
seg_a:0654	f,	es=0		
seg_a:0987	f,	od		

Приложение

		seg_a:0987 непосредственный операнд будет представлен как ссылка на данные относительно регистра cs (умолчание).
seg_a:0987 f,	od es	В команде по адресу seg_a:0987 непосредственный операнд будет представлен как ссылка на данные относительно регистра es.
seg_a:0121 f,	sn seg_c:0032 4	Команда по адресу seg_a:0121 использует таблицу вызовов подпрограмм, расположенную по адресу seg_c:0032 и имеющую 4 элемента.

Формат файла комментариев

С первой позиции номер комментария (0-3FFh). Далее произвольное количество строк текста с пробелами в первой позиции. Заканчивается комментарий началом следующего комментария или словом end с первой позиции.

Пример файла комментариев

```
001 Комментарий номер один.
002 Комментарий номер 2:
      – строка 1
      – строка 2
      – строка 3
003 Комментарий номер 3:
      – строка
      – строка 2
      – строка 3
end
004 Комментарий номер 4.
```

Формат файла определения комментариев к оборудованию (HARDWARE.TBL)

Для запуска SOURCER с файлом определения комментариев, находящимся не в текущей директории и/или имеющим имя, отличное от HARDWARE.TBL, в командной строке (ключ -h) или с помощью SET переменной VCOM необходимо задать имя файла определения комментариев к оборудованию:

```
set vcom=hd:\sourcer\example.tbl
```

Все команды в этом файле начинаются с первой позиции. Если в первой позиции стоит пробел, то такая строка считается строкой комментария.
Комментарий следует за командой, состоящей из поля типа, диапазона (области действия), действия.
Текст комментария начинается после точки с запятой и пробела.
Длина комментария к прерыванию ограничена 30 символами, а к порту ввода-вывода восемнадцатью.

Формат команды:

types	range	actions	комментарий
тип	диапазон	действия	

Приложение

```
Types: int  — комментарий к прерыванию.
       in   — комментарий к порту ввода.
       out  — комментарий к порту вывода.
       io   — комментарий к порту ввода-вывода.
Range: all  — для удаления всех типов комментариев.
       ndos — для удаления всех комментариев
              к нестандартным DOS прерываниям
              (30h и больше).
       bios — для удаления всех комментариев
              к прерываниям BIOS в диапазоне
              от int 5h до int 1Ch.
       #h   — конкретное шестнадцатиричное значение
              (# представлят число).
Actions: delete  — удалить комментарий (только для int и io).
         -blank- — однострочный комментарий.
         ax=func — первая строка многовариантного
                   комментария к прерыванию,
                   значение ax выводится в листинг
                   со смещением 26 символов от начала.
         ah=func — первая строка многовариантного
                   комментария к прерыванию,
                   значение ax выводится в листинг
                   со смещением 28 символов от начала.
         ax=#    — значение регистра ax имитатора
                   должно быть равно
                   значению # (только int).
         ah=#    — значение регистра ah имитатора
                   должно быть равно
                   значению # (только int).
         al=#    — значение регистра al имитатора
                   должно быть равно значению #
                   (только int).
```

Примеры:

```
type range actions comments
---- ----- ------- --------
int  all,   delete  (Удаляются все комментарии к прерываниям)
int  ndos,  delete  (Удаляются нестандартные DOS прерывания)
int  bios,  delete  (Удаляются прерывания BIOS в интервале
                     от int 5h до int 1Ch)
int  1Ah,   delete  (Удаляется комментарий к int 1A и все
                     комментарии к подфункциям)

int  9,             ; keyboard
int  10h,   ah=func ; Video Services ah=func  xxh
int  10h,   ax=101Ah; read color page state
int  10h,   ah=15h  ; get active display parameters

io   all,   delete  (Удаляются все комментарии к портам в/в)
io   37Ah,  delete  (Удаляется комментарий к порту в/в 37Ah)
io   64h,   al=ADh  ; disable keyboard
out  0F1h,          ; 80287 reset
io   278h,          ; printer-3 data
in   321h,          ; controller status
out  321h,          ; controller reset
```

Имеется ограниченное количество свободных элементов в таблице комментариев, поэтому может потребоваться удаление ненужных комментариев при добавлении собственных комментариев.
Для справки приводим количество элементов, занимаемых в таблице комментариев некоторыми программными прерываниями с их подфункциями.

int	10h	Видео BIOS	100
int	20h	Windows VxD функции	350
int	33h	Функции "мыши"	50
int	67h	EMS и VCPI	70
io	64h	Контроллер клавиатуры	20
io	70h	RTC/CMOS регистры	30
io	3B4h	Видеоконтроллер	20
io	3D4h	Видеоконтроллер	20

■

Двухкомпонентная антивирусная система

Ю. Косивцов

Преимущества резидентных "сторожей"

Бороться с вирусами можно различными способами. Чаще всего используются детекторы вирусов, которые по известной сигнатуре находят вирус в зараженном файле. Как правило, используют AIDSTEST Д.Н. Лозинского, притом не самых последних версий. AIDSTEST — это прекрасная программа, но довольно часто возникают случаи, когда она не может помочь. Это происходит, когда Ваш компьютер поражен вирусом, который неизвестен программе Д.Н.Лозинского или ей подобной. Я работаю в политехническом институте, и мне довольно часто приходится иметь дело со студенческими самоделками. Может быть, такова человеческая природа, но, едва научившись нажимать на клавиши и узнав несколько команд ассемблера, многие студенты задаются целью написать вирус. Чаще всего подобные вирусы-самоделки крайне примитивны и представляют собой разновидности Венского вируса. Однако их опасность от этого не уменьшается. Во-первых, они не распознаются AIDSTEST и подобными программами, а если и распознаются по знакомым сигнатурам, то лечатся неправильно. Во-вторых, почти в каждом запланировано вредоносное действие. В этой области фантазия безгранична: от простых попыток форматирования винчестера до рисования на экране различных рисунков. Один раз пришлось иметь дело с редкостным экземпляром, который представлял собой примитивный резидентный вирус, но вред от него был очень своеобразен. Автор этого вируса экспериментировал с прерыванием таймера (08h) и портами 3D4h и 3D5h контроллера видеотерминала 6845 адаптера CGA. Примерно через 10 минут после активизации вируса на мониторе появлялись полосы кадровой развертки. Еще через 10 минут их бег ускорялся. Представьте Вашу реакцию, скорее всего, она будет следующей: "Сломался монитор", — и в лучшем случае Вы ограничитесь покручиванием ручки кадровой развертки и постукиванием кулаком по корпусу монитора, но вряд ли подумаете, что это действие вируса.

Для борьбы с "незнакомцами" предназначены антивирусы, работающие по другому принципу: они считывают информацию с диска с последующим ее сравнением с эталоном. Если произошли незапланированные изменения, это значит, что у Вас завелся вирус. Наиболее популярен среди ревизоров дисков ADINF Д.Ю.Мостового и В.С.Ладыгина. ADINF — это, по моему мнению, лучшая антивирусная программа из ныне существующих. Технология поиска вирусов, примененная в ADINF, обеспечивает обнаружение всех типов вирусов, включая так называемые стелсы, маскирующиеся под часть операционной системы. Стелс — это вирус, который встраивается в ядро операционной системы и перехватывает обращения к драйверу дисков DOS. При попытке чтения зараженного файла через функции 21-го прерывания, а иногда и через 25-е прерывание подставляют неверную информацию, так что файл выглядит как незараженный. Однако при чтении секторов, принадлежащих файлу, через INT 13h вирус оказывается на месте. Работа ADINF основана на сравнении информации, полученной чтением через INT 13h и INT 25h. Если имеется различие, Вас можно поздравить: Вы имеете в своей коллекции стелс. Такая методика позволяет выявить практически все существующие вирусы, а также вирусы, которые будут появляться в будущем. Теоретически, конечно, можно обмануть и ADINF, используя сле-

дующую методику: вирус загружается резидентно и анализирует частоту изменения EXE файлов. Ведь практически на каждом компьютере кто-нибудь что-нибудь да программирует. Наверняка имеются компиляторы TURBO, QC, QB, CLIPPER и тому подобные. В процессе написания программ создаются и изменяются EXE файлы. Так что, если в результате анализа выявлено, что EXE файл часто изменяется и имя изменяющей программы — одно из перечисленных, то это наверняка работа компилятора. При очередном изменении можно добавить в EXE файл коды вируса и есть шанс, что Ваш заказчик получит уже зараженную программу. ADINF в этом случае, конечно, покажет изменения на диске, другое дело, обратите ли Вы на них внимание. Но подобные технологии достаточно сложны в исполнении и вряд ли когда-нибудь будут реализованы, и ADINF на сегодняшний день остается лучшей антивирусной программой, выявляющей 100% вирусов.

Однако при всех достоинствах ревизоров у них есть один существенный недостаток. Они могут только показать наличие вирусов, но "вылечиться" от них — это Ваша проблема. Если в вашей организации есть квалифицированные программисты, то проблема отпадает. Если Вы живете в большом городе, где всегда есть к кому обратиться за помощью, то это тоже полбеды. Но если нет ни того ни другого, тут, увы, ADINF Вам не поможет. Я один раз был в командировке в маленьком поселке, где, однако, находится большой химкомбинат и довольно много компьютеров. Настоящим стихийным бедствием там был печально известный DIR. Его пытались уничтожить с помощью ANTI-DIR Д.Н.Лозинского, однако почему-то запускали его без ключа /F (лечить файлы). Начальник одного из цехов этого химкомбината жаловался: "Я его лечу, лечу, а он опять откуда-то вылезает". В результате небольшого исследования DIR был обнаружен мной даже на заклеенной системной дискете! Уважаемые читатели, как Вы думаете, нужен ли на этом химкомбинате ADINF? Помочь в этом случае могут резидентные "сторожа" или фильтры, которые встраиваются в цепочку прерываний и анализируют обращения к INT 13h или INT 21h. При обнаружении опасных действий, например, записи в COM файл "сторож" может выдать предупреждающее сообщение, а иногда прервать выполнение программы. Это позволяет обнаружить вирус в самом начале его распространения, когда заражены 1-2 файла и их можно просто удалить. Фильтры тоже не лишены недостатков, самый существенный из которых то, что они очень мешают работать, но не являются препятствием для вирусов. Этого, правда, нельзя сказать про фильтр А.А.Чижова, который проверяет контрольную сумму блоков памяти после завершения программы и выдает сообщение в случае ее изменения. Такой подход почти полностью исключает, что вирусу удастся незаметно остаться в оперативной памяти, а обойти подобную проверку достаточно сложно. Большинство же авторов фильтров считают, что если они перехватили, например, INT 27h, то безопасность гарантирована. Существует несколько вариантов обхода подобной "защиты", многие были описаны Е.Касперским ("Дыры" в MS-DOS и программы защиты информации// КомпьютерПресс.– 1991.– № 10), и я их перечислю, но на некоторых хотелось бы остановиться подробнее.

"Защита" INT 13h и ее обход

Все программы перехвата INT 13h можно свести к следующему (пример 1). Представленная программа запретит запись на жесткий диск. Подобными программами изобилует "Софтпанорама", они могут быть несколько сложнее, например, могут запрещать запись только в определенные секторы или проверять, пришел ли запрос на запись из драйвера дисков DOS, но суть их остается такой же. Пускай извинят меня их авторы, но толку от такой "защиты" нет никакого. Прежде всего, все такие "сторожа" ориентированы не на пресечение распространения вирусов, кроме, может быть, бутовых, а на исключение вреда от них. Это абсолютно не имеет никакого смысла, так как вирусы причиняют вред не только винчестеру, а кроме того, для доступа к винчестеру совсем не обязательно пользоваться 13-м прерыванием. Во-вторых, достаточно просто получить адрес оригинального обработчика INT 13h. Способов получения этого адреса мне известно четыре.

Первый. Получение адреса использованием 13-й функции прерывания 2Fh. Способ был неоднократно описан, и я не буду на нем останавливаться. Теоретически на пути вируса, использующего этот способ, можно поставить препятствие в виде обработчика INT 2Fh, который выявит использование функции 13h.

Пример 1

```
; Пример абсолютно бесполезной программы перехвата
; 13-го прерывания, которая якобы запретит вирусу
; запись на жесткий диск
CODE    SEGMENT
        ASSUME  cs:CODE, ds:CODE
        org     100h
START:
        jmp     short INIT  ; Переход на процедуру инициализации

INT_13_ENTRY    PROC FAR ; Обработчик 13-го прерывания

        sti                 ; Разрешить прерывания
        cmp  ah,3           ; Функция записи секторов
        jne  LOC_1          ; Если нет, то ничего не делать
        cmp  dl,80h         ; Запись производится на жесткий диск ?
        jb   LOC_1          ; Если нет, то ничего не делать,
        mov  ah,3           ; иначе код ошибки
                            ; "Устройство защищено от записи"
        stc                 ; Взвести CARRY-флаг
        ret  2              ; Возврат в вызывающую программу

LOC_1:                      ; Сюда передается управление
                            ; при "безобидных" функциях
        cli                 ; Запретить прерывания
        DB   0EAh           ; Инструкция JMP FAR на оригинальный
IP_13   DW   ?              ; обработчик 13-го прерывания
CS_13   DW   ?              ;

INT_13_ENTRY    ENDP        ; Конец обработчика 13-го прерывания

INIT    PROC NEAR           ; Начало процедуры инициализации

        mov  ds,ax          ; ds = ax = 0
        lds  ax,ds:[4Ch]    ; ds:ax - адрес обработчика 13-го прерывания
        mov  es:CS_13,ds        ; Сохранение адреса обработчика
        mov  es:IP_13,ax        ; и построение инструкции JMP FAR на него
        mov  ds,bx          ; ds = bx = 0
        cli                 ; Запретить прерывания
        mov  ds:[4Eh],cs    ; Занести в таблицу векторов прерываний адрес
        mov  word ptr ds:[4Ch], OFFSET INT_13_ENTRY   ; процедуры
                                                      ; обработки
        sti                 ; Разрешить прерывания
        mov  dx, OFFSET INIT ; Смещение первого байта
                             ; за резидентной частью
        int  27h            ; Завершить и остаться резидентным

INIT    ENDP

CODE    ENDS
        END START
```

Второй. Трассировка — тоже достаточно известный способ. Вирус должен иметь процедуру, аналогичную приведенной в примере 2, выставлять на нее вектор 1-го прерывания, затем загрузить в регистр AH какой-нибудь безобидный номер функции (например, проверки секторов), установить флаг трассировки и выполнить CALL FAR для обработчика INT 13h. Обработчик 1-го прерывания анализирует сегментный адрес, откуда пришел вызов, и если он становится больше или равен 0C800h, значит, точка входа найдена. Трассировщик должен анализировать следующую за трассируемой команду и команды загрузки сегментных регистров, которыми можно замаскировать другие команды на предмет возможности сброса флага трассировки или анализа выполнения с трассировкой (CLI, PUSHF, POPF, INT, INTO, INT 3) и команды загрузки сегментных регистров, которыми можно замаскировать другие команды. Фрагмент кода, представленный в примере 3, испортит работу трассировщика, не имеющего такого анализатора.

Кстати, этот фрагмент, включенный в Вашу программу, может испортить настроение хэкеру, который пытается посмотреть Вашу программу отладчиком. Добавьте вместо точек килобайтов 20 кода, который абсолютно ничего делать не будет, и пусть он с ним разбирается. Не войти в обработчик 60-го прерывания он не сможет, так как тогда отладчик не получит управления, а если войдет, то очень долго не выйдет.

Выставленный флаг трассировки можно выявить косвенно, замаскировав аппаратные прерывания, поместив в [SP-1] контрольное значение и дав инструкцию STI. Тогда по изменению слова в стеке можно судить о том, было трассировочное прерывание или нет. Есть и другие способы узнать это (конвейер команд и тому подобное). Однако использование подобных методов существенно увеличивает размер резидентного кода, а обнаружение трассировки методом конвейера команд ведет, как правило, к сбою системы. Кроме того, вирусу не обязательно пытаться трассировать INT 13h, можно трассировать INT 2Fh и затем его использовать для получения адреса INT 13h. Как говорится: "Знал бы, где упадешь, соломы подстелил бы". Будем считать защиту от этого способа получения адреса INT 13h условно возможной.

Третий. Метод фиксированных адресов. В настоящее время на большинстве ПК используется MS-DOS версий 3.30, 4.01 и 5.00. Вирус после идентификации версии DOS может обратиться по адресам, специфическим для нее, где хранится оригинальный адрес обработчика INT 13h. Метод может не работать, так как наряду с MS-DOS часто используются DR DOS, COMPAQ DOS и другие, однако в случае успешного применения защита от вируса срабатывает.

Четвертый. Поиск и анализ драйвера винчестера BIOS. Этот метод нигде не был опубликован, но применялся в вирусе DIR, поэтому я остановлюсь на нем подробнее.

Абсолютное большинство компьютеров оснащены жесткими дисками. Жесткий диск, однако, является дополнительным оборудованием, и его BIOS располагается в так называемом дополнительном сегменте BIOS с интервалом сегментных адресов 0C000h-0F000h. Драйверы устройств, расположенные в дополнительном сегменте BIOS, должны отвечать определенным требованиям: начинаться с адреса, кратного 2K, и иметь заголовок, представленный на рис. 1, для того чтобы основной сегмент BIOS мог найти их при загрузке компьютера.

Пример 2

```
; Пример трассировщика 13-го прерывания

INT_1   PROC NEAR       ; Обработчик 1-го прерывания
        push bp         ; Сохранить BP
        mov  bp,sp      ; Загрузить в BP указатель стека
        cmp  word ptr ss:[bp+4],0C000h ; Проверка сегмента
        jae  _EXIT      ; Если сегмент BIOS, то конец трассировки
; Далее должен быть анализатор "опасных" команд
        ...
        pop  BP         ; Восстановление BP
        iret            ; Возврат (продолжение трассировки)

_EXIT:  mov  ax,[bp+4]  ; В AX сегмент INT 13h
        mov  bx,[bp+2]  ; В BX смещение INT 13h
        jmp  _OK_TRACE ; Выход из обработчика (конец трассировки)

INT_1   ENDP            ; Конец обработчика 1-го прерывания

; Пример вызова трассировщика 13-го прерывания
...
        push ds         ; Сохранить DS
        mov  SAVE_SS,ss      ; Сохранить состояние стека в переменных
        mov  SAVE_SP,sp      ;
        pushf           ; Флаги в стек для эмуляции инструкции INT
        pushf           ; Флаги в стек для взведения флага трассировки
        pop  ax         ; Флаги в AX
        or   ax,100h    ; Выставление 8-го бита
        push ax         ; Запись AX в стек
        xor  ax,ax      ; Обнуление AX
        mov  ds,ax      ; Обнуление DS
        mov  ax,0401H   ; Проверять один сектор
        mov  cx,1       ; Дорожка 0, сектор 1
        mov  dx,80h     ; Сторона 0, устройство 80h
        mov  bx,OFFSET BUFFER ; Буфер на всякий случай
        push cs         ; Выставление ES на буфер
        pop  es         ;
        popf            ; Взведение флага трассировки
        call dword ptr ds:[4CH] ; CALL FAR на обработчик INT 13h
        pop  ds         ; Если вернулось управление, значит флаг
        jmp  _ERROR_TRACE ; трассировки был сброшен, ошибка
_OK_TRACE:              ; Сюда управление передается в случае удачи
        mov  dx,cs:SAVE_SS ; Загрузка DX сохраненным SS
        cli             ; Запретить прерывания
        mov  ss,dx      ; Восстановить стек из переменных
        mov  sp,cs:SAVE_SP ;
        sti             ; Разрешить прерывания
        pop  ds         ; Восстановить DS
; Далее можно использовать полученное значение точки входа INT 13h,
; которое содержится в AX:BX
        ...
```

Пример 3

```
INT_60 PROC FAR ; Обработчик 60-го прерывания
        sti     ; Разрешить прерывания
        push bp ; Сохранить BP
        mov  bp,sp ; Установить BP на вершину стека
        ...
        add  [bp+2],1  ; Изменить в стеке адрес возврата
        ...
        pop  bp ; Восстановить BP
        retf 2  ; Возврат из прерывания со сбросом
                ; флага трассировки
INT_60 ENDP             ; Конец обработчика 60-го прерывания

; Участок кода, вызывающий обработчик 60-го прерывания
        ...
        pop  es ; Загрузка сегмента регистра для маскирования
        int  60h ; Вызов прерывания
        nop     ; Сюда должно вернуться управление
        nop     ; Сюда управление вернется
        ...
```

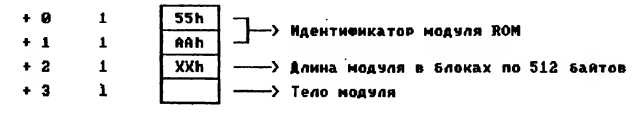


Рис. 1

Программа инициализации BIOS производит инициализацию устройства и выставляет вектор INT 13h на нужный адрес. Установку вектора проще всего произвести командой MOV, команды STOSW и MOVSW встречаются чрезвычайно редко.

Можно использовать следующую схему поиска точки входа INT 13h. Сначала ищут драйвер дисков в дополнительном сегменте BIOS, начиная с сегмента 0C000h по сигнатуре 55AAh (WORD 0AA55h). Затем в теле драйвера ищут сигнатуру C7 06 4C 00, соответствующую команде

```
MOV WORD PTR [4CH],OFFSET INT_13_ENTRY_POINT
```

Когда сигнатура найдена, остается только прочитать слово по смещению +4 от начала сигнатуры (то, что эта команда записывает по данному адресу). Это и будет смещение точки входа обработчика INT 13h, а сегмент точки входа равен сегменту драйвера.

Такая схема поиска оригинального обработчика обеспечивает почти полную совместимость с любым BIOS и полностью не зависит от операционной системы. Защита от нее на программном уровне невозможна, она полностью пробивает все возможные "защиты", основанные на перехвате 13-го прерывания.

Из изложенного можно сделать вывод: почти все системы защиты, основанные на перехвате 13-го прерывания, абсолютно бесполезны. Приведенные методы "взлома защиты" 13-го прерывания — это не пособие авторам вирусов, а предупреждение людям, пытающимся защитить свой винчестер от посягательств, не применять подобные "защиты". Для "Софтпанорамы" их публикации простительны (все-таки вы получаете их бесплатно), но часто встречаются случаи, когда подобные системы продаются или разрабатываются на заказ за приличное вознаграждение. Авторы подобных систем или люди, не компетентные в своей области, или просто жулики. Всегда интересуйтесь, по какому принципу работает устанавливаемая Вами защита, так как защита 13-го прерывания без применения специальных аппаратных средств невозможна.

Защита 21-го прерывания

Наиболее узким местом вирусов является проникновение в систему. Именно в это время вирус наиболее уязвим и правильно организованный "сторож" может его засечь. Для внедрения в систему вирусам (кроме бутовых) необходим доступ к функциям 21-го прерывания. Для обхода возможных "сторожей" вирусы должны снимать защиту с 21-го прерывания. Методов "взлома" защиты INT 21h существует больше, чем защиты INT 13h. Они приведены у Касперского, и я не буду на них останавливаться. Полностью совместимым со всеми версиями DOS можно считать только способ трассировки. Несмотря на это, я сильно сомневаюсь в эффективности систем защиты 21-го прерывания, основанных на встраивании в цепочку прерывания, а именно так чаще всего и организуется защита.

Однако защиту 21-го прерывания можно организовать более эффективно, используя метод встраивания в ядро операционной системы. Обычно это делается по следующей схеме: в точку входа INT 21h записывается инструкция JMP FAR для обработчика, который проверяет номер функции на безопасность, потом восстанавливает на нее точку входа INT 21h и делает CALL FAR на нее. После возврата управления в точку входа DOS опять записывается инструкция JMP FAR и управление передается программе, которая вызвала INT 21h.

Данный подход имеет ряд существенных недостатков. Во-первых, значительно увеличивается время вызова INT 21h, что ощутимо уменьшает быстродействие практически любого программного обеспечения. Во-вторых, вызов INT 21h по CALL FAR несет в себе возможность сбоев системы из-за лишних трех слов в стеке (флаги, cs, ip). В-третьих, подобный обработчик должен быть достаточно сложным из-за необходимости выявлять особые случаи (функции с номерами 0, 4Bh, 4Ch и так далее) и обрабатывать внутренние вызовы DOS, что, в свою очередь, приводит к существенному увеличению его размера и еще больше снижает быстродействие системы.

Для DOS версий 3.xx и 4.xx подобную программу можно несколько улучшить, предусмотрев в ней обработку случая превышения максимально допустимого для данной версии номера функции. Тогда команда дальнего перехода на обработчик точно впишется в 5 байтов первой команды DOS, которая делает подобную проверку. Это позволит избежать восстановления точки входа DOS и передать управление системе не инструкцией CALL FAR, а инструкцией JMP FAR, что, в свою очередь, снимет большинство возникающих проблем. Но такой подход неприменим в случае DOS версии 5.00, и, кроме того, в вирусе достаточно просто запрограммировать анализатор того, является ли первая инструкция в сегменте DOS ее истинной точкой входа или это JMP FAR. В этом случае вирус может передать управление не на точку входа, а, например, по смещению +7 от нее для DOS версий 3.xx и 4.xx (пример 4).

С первого взгляда на листинги точек входа DOS различных версий бросается в глаза большое количество трехбайтовых инструкций (CMP AH,...). Они идеально подходят для замещения их на трехбайтовый ближний переход (JMP NEAR). Использование ближних переходов устраняет все осложнения, связанные со

Пример 4

```
; Листинги точек входа INT 21h

; DOS версий 3.x и 4.x (подробнее см. в статье Касперского)

2E: 3A 26 .. ..  CMP  AH,CS:DOS_MAX_FUNC_N ; Проверка допустимости
                                            ; номера функции
77 ..            JA   ERROR_NO      ; Номер больше допустимого, ошибка
80 FC 51         CMP  AH,51H        ; Номер функции 51h ?
74 ..            JE   FUNC_51       ; Да, переход на обработчик
80 FC 62         CMP  AH,62H        ; Номер функции 62h ?
74 ..            JE   FUNC_62       ; Да, переход на обработчик
                 ...                ; и т.д.

; DOS версии 5.00 при загрузке в младшие адреса памяти

FA               CLI                ; Запретить прерывания
80 FC 6C         CMP  AH,6CH        ; Проверка допустимости
                                    ; номера функции
77 ..            JA   ERROR_NO      ; Номер больше допустимого, ошибка
                 ...                ; и т.д.

; DOS версии 5.00 при загрузке в старшие адреса памяти

90               NOP                ; Эти две команды выставляются при
90               NOP                ; загрузке драйвера HIMEM.SYS
E8 .. ..         CALL .. ..         ; Проверка включения
                                    ; адресной шины A20
2E FF 2E .. ..   JMP  FAR CS:[.. ..] ; Переход на обработчик DOS
                 ...
```

встраиванием в ядро операционной системы. Чтобы их использование стало возможным, адрес "сторожа" должен отличаться от адреса ядра DOS не далее чем на 32 килобайта. Этого можно достичь, использовав в качестве "сторожа" системный драйвер.

Организация защиты 21-го прерывания путем встраивания в ядро операционной системы позволяет также провести очень существенную проверку — проход цепочки прерывания.Прежде всего хочу отметить следующую особенность. Большая часть обычного программного обеспечения вызывает 21-е прерывание инструкцией INT 21h, в то время как вирусы обычно делают CALL FAR или JMP FAR на точку входа для обхода возможных резидентных "сторожей" (минимальный стелс механизм). Исключения из этого правила бывают, когда критично время вызова 21-го прерывания или в условиях специальной отладки, и этих исключений очень немного. Это обстоятельство и надо использовать для выявления вирусов путем использования двухкомпонентной антивирусной системы. Первый компонент встраивается в ядро DOS, а второй просто перехватывает цепочку 21-го прерывания. Когда программа выполняет инструкцию INT 21h, управление передается второму компоненту. Он может сделать проверки на опасность функции, затем выставить переменную "проход цепочки" и передать управление дальше. При получении управления первым компонентом он проверяет переменную "проход цепочки". Если она выставлена, то была инструкция INT 21h, надо сбросить переменную "проход цепочки" и передать управление DOS. Если переменная сброшена, то вызов пришел напрямую и надо принимать меры: скорее всего, это действие вируса.

Кроме того, "сторож" обязательно должен делать следующие проверки: вызов функции 3Dh для открытия файлов с расширениями COM, EXE, SYS, BIN, во избежание записи вирусом в них своих кодов, вызов функции 52h, вызов функции 5Dh, сканирование блоков памяти при завершении процесса. Комплекс этих проверок делает практически невозможным внедрение вирусов в систему. Совершенно очевидно, что эти проверки должен осуществлять второй компонент, так как срабатывание первого компонента (обход цепочки прерывания) уже является признаком попытки внедрения вируса. Не надо делать проверки на использование 31h функции — это бесполезно, так как завершиться с сохранением резидентных свойств можно и без нее (например, функцией 4Ch, несколько изменив свой MCB). Абсолютно бессмысленно проверять запись секторов (чаще всего это действия уже внедрившегося вируса). Чем проще "сторож" и чем короче его код, тем более эффективно он будет работать.

Все изложенные выше концепции были проверены и нашли свое отражение в моем антивирусном "стороже". Листинг рабочего варианта на ассемблере для DOS версий 3.xx, 4.xx и 5.00 будет помещен в электронной версии журнала. ■

езидентный перехватчик функции EXEC

Наверное не будет преувеличением сказать, что при работе на ПЭВМ коллективного пользования типична ситуация, когда каждый пользователь чувствует себя профессионалом и поэтому думает, что он вправе приносить и запускать все что угодно.

Последствия самые печальные. Это и утраченные данные, сброшенные CMOS, испорченные программы и прочее. О вирусах я и не говорю, и так ясно. Причем, что интересно, ежели случились неприятности, то Вы не всегда сможете добиться от пользователя вразумительного ответа на вопрос: "Что запускалось на этой машине?". Он, видите ли, не помнит или даже не знает, что он запускал...

А вот еще одна ситуация: иногда необходимо собрать статистические данные о характере решаемых задач, о степени использования программного обеспечения, установленного на ПЭВМ. Как это сделать, не обращаясь к пользователю?

Очевидно, что надо установить какую-нибудь резидентную программу, которая незаметно для пользователя фиксировала бы его действия. Ну что же, комплексом шпиономании мы не страдаем, поэтому будем фиксировать только выполнение функции EXEC (загрузить и выполнить программу). Для этого напишем соответствующий обработчик прерывания int 21h и заменим в таблице векторов адрес процедуры прерывания 21h на наш. В результате мы имеем скрытый файл-протокол, содержащий список выполняемых задач с переданными им аргументами.

Почему нас интересует только функция EXEC? Потому, что наша программа ориентирована на DOS, а как известно, COMMAND.COM и другие запускают программу на выполнение именно этой функцией, и было бы логично в этот момент выполнять все свои "черные" дела. К тому же, в нашу задачу не входит создание чуда TSR, которое займет несколько килобайтов ОЗУ. Структура вызова функции EXEC довольно проста. Для тех, кто не знаком с ней, приведем краткое ее описание. Это поможет вам при изучении текста программы понять проводимые в ней манипуляции с регистрами.

Функция EXEC — загрузить и выполнить программу

Вызов: AX = 4B00h

DS:DX — указатель на строку ASCIIZ, содержащую дисковод и маршрут выполняемого файла
ES:BX — указатель на блок параметров, состоящий из 4 адресов:

СМЕЩЕНИЕ	ДЛИНА	ОПИСАНИЕ
00h	WORD	Сегментный адрес среды
02h	DWORD	Указатель на командную строку
06h	DWORD	Указатель на первый FCB
0Ah	DWORD	Указатель на второй FCB

Возвращает: CF сброшен — нет ошибок,
CF установлен — произошла ошибка (код ошибки в AX).

Программа Load$.COM, как это и положено TSR программе, стандартным образом устанавливает свою резидентную часть в оперативную память, а при запуске с ключом /END выгружается из памяти и восстанавливает захваченные векторы прерываний. Текст программы приводится с подробными комментариями, так что читатель без труда найдет соответствующие участки.

Любителям модификаций можно посоветовать заняться сбором дополнительной информации: например, можно фиксировать копию среды родительской программы (что-то типа verify = on) или определять и заносить в файл дату и время запуска задачи. Для этого надо вставить соответствующую процедуру в обработчик прерывания. А любителям фиксировать загрузку оверлейных файлов придется самостоятельно отслеживать функцию AX=4B03h/21h. Впрочем, что надо, а что не надо, Вам подскажут Ваши потребности и Ваша фантазия.

Удачи!

■

Листинг

```
;-----------------------------------------------------
; Load$.ASM was written by AVA Soft.
; Copyright (c) 1992
; Программа может быть свободно использована
; в некоммерческих целях
; Автор: Аброськин Андрей Владимирович
;-----------------------------------------------------

code    segment
        assume  cs:code,ds:code,es:code
        org     100h

start:  jmp     install     ; Инициализация программы
                            ;
presence dw     0427h       ; Ключевое слово присутствия
                            ; программы в памяти

int21_entry   proc    far   ; Наш обработчик
                            ; прерывания 21H
        pushf
        push    ax
        push    bx
        push    cx
        push    dx
        push    di
        push    si
        push    bp
        push    es
        push    ds
        mov     cs:@ds,ds       ; Сохраняем регистры для
        mov     cs:@dx,dx       ; последующих маневров
        mov     cs:@es,es
        mov     cs:@bx,bx
        cmp     ax,4B00h        ; Это функция Exec?
        je      short Exec      ; Да - работаем с ней,
        jmp     return          ; нет - выходим из
                                ; обработчика
Exec:
        cld                     ; Загружаем ES,DI
        mov     es,cs:@DS       ; указатель DS,DX
        mov     di,cs:@DX       ; на ASCIIZ строку
        mov     cx,0F0h         ; загружаемой программы
        xor     ax,ax
        repnz   scasb           ; Сканируем в поисках нуля
        neg     cx              ; и вычисляем длину строки
        add     cx,0F0h         ;
        mov     cs:@CX,cx       ; Сохраняем полученный результат
        mov     ds,cs:@DS       ; Загружаем DS,SI
        mov     si,cs:@DX
        push    cs
        pop     es              ; Настраиваем Buffer на смещение 02h
        lea     di,cs:[Buffer+2]
        rep     movsb           ; Пересылаем ASCIIZ строку в Buffer
        mov     ds,cs:@ES       ; Загружаем в DS,SI
        mov     si,cs:@BX       ; указатель ES,BX на блок параметров
        lodsw                   ; Пропускаем сегментный адрес среды
        lodsw                   ; Вычисляем через регистр AX сегмент
        mov     cs:@BX,ax       ; и смещение указателя на строку
        lodsw                   ; передаваемых аргументов
        mov     ds,ax           ; Загружаем DS,SI вычеленными
        mov     si,cs:@BX       ; значениями
        lodsb                   ; Пересылаем в AL число нажатий на
        cmp     al,1            ; клавиатуру (длину аргументов),
        jbe     exit            ; если параметров нет - прекращаем
        cbw
        mov     cx,ax           ; Настраиваем CX на длину параметров
        mov     cs:@DS,ax       ; и временно сохраняем значение
        mov     ds,cs:@ES       ; Загружаем DS,SI вычисленным значением
        mov     si,cs:@BX       ; Корректируем регистр SI
        inc     si              ; Настраиваем ES,DI для засылки
        mov     ax,cs           ; строки аргументов в Buffer
        mov     es,ax
        lea     di,cs:[Buffer+2]
        add     di,cs:@CX
        rep     movsb           ; Пересылаем то, что хотели
        mov     ax,cs:@DS       ; Не забываем подправить счетчик
        add     cs:@CX,ax       ;
                                ; Окончательно корректируем счетчик
exit:   add     cs:@CX,2        ; и переходим к работе с файлом

        mov     ah,62h          ; Вычисляем адрес PSP
                                ; прерванной программы
        pushf
        call    dword ptr cs:[old21o]

        mov     cs:PPSP,bx      ; Сохраняем PSP текущего процесса,
        mov     bx,cs:TPSP      ; и заменяем его на "наш" PSP
        mov     ah,50h          ; Устанавливаем новый адрес PSP
        pushf
        call    dword ptr cs:[old21o]

                                ; PUSHF
        mov     ax,cs           ; CALL DWORD PTR CS:[old21o] -
        mov     ds,ax           ; подпольный вызов int 21h
        mov     dx,offset File;
        mov     ax,3D01h        ; Открываем файл для записи
        pushf
        call    dword ptr cs:[old21o]

        jc      open_error      ; В случае ошибки
                                ; пытаемся его создать
open:                           ;
                                ; Сохраняем handle файла
        mov     bx,ax           ; в регистре BX
        mov     ah,42h          ;
        mov     al,2            ; Передвигаем указатель
        xor     cx,cx           ; в конец файла
        mov     dx,cx           ;
                                ;
        pushf
        call    dword ptr cs:[old21o]

        mov     ax,cs           ; Загружаем DS через CS,
        mov     ds,ax           ; а в CX занесем вычисленную
        lea     dx,cs:[Buffer]  ; ранее длину записываемой строки
        mov     cx,cs:@CX       ;
        mov     ah,40h          ; Пишем Buffer в File
        pushf
        call    dword ptr cs:[old21o]
                                ;
close:                          ;
        mov     ah,3Eh          ; Закрываем файл
        pushf
        call    dword ptr cs:[old21o]

        jmp     short exit_21H ; Умываем руки и готовимся
                                ; отдать управление "настоящему"
open_error:                     ; обработчику прерывания 21h
        mov     ax,cs
        mov     ds,ax
        mov     dx,offset File  ; Создаем file handle
        mov     cx,02h          ; для File с атрибутом
        mov     ah,3Ch          ; "скрытый" (CX = 02h)
        pushf
        call    dword ptr cs:[old21o]

        jc      short exit_21H ; Выходим по ошибке
        jmp     short open      ; Переходим к работе с файлом
                                ;
                                ;
exit_21H:
        mov     bx,cs:PPSP      ; Восстанавливаем адрес PSP
        mov     ah,50h          ; прерванной программы
        pushf
        call    dword ptr cs:[old21o]

return:
        pop     ds              ;
        pop     es              ;
        pop     bp              ;
        pop     si              ;
        pop     di              ; Восстанавливаем регистры
        pop     dx              ;
        pop     cx              ;
        pop     bx              ;
        pop     ax              ;
        popf

        db      0EAh            ; Инструкция far jump на
old21o  dw      0               ; старый адрес прерывания 21H -
old21s  dw      0               ; смещение и сегмент

int21_entry     endp
file    db      "c:\load$.tmp",0
                                ; Область данных:
Buffer  db      0Dh,0Ah         ; буфер для хранения строки
        db      80 dup (20h)    ; (первые 2 байта - клавиша <ENTER>)
@DS     dw      (?)             ;
@DX     dw      (?)             ; Переменные для хранения
@ES     dw      (?)             ; совершенно секретных данных
@BX     dw      (?)             ;
```

Листинг (продолжение)

```
@CX         dw      (?)         ; Счетчик (число записываемых байтов)
PPSP        dw      (?)         ; PSP прерванной программы
TPSP        dw      (?)         ; PSP резидентной программы

;-----------------------------------------
; Транзитная часть программы  Load$.COM
;-----------------------------------------

install:                ;
        mov     ax,3521h        ; Получаем адрес обработчика
        int     21h             ; прерывания 21H
        mov     ax,es:[bx-2]    ; Прерываем ключевое слово
        cmp     ax,cs:presence  ; присутствия программы в памяти
        jnz     load            ; Нет - переход на load
        jmp     already         ; Да - переход на already

load:
        push    es              ; Сохраняем регистры
        push    ds              ;
        mov     ah,62h
        int     21h
        mov     cs:TPSP,bx      ; Вычисляем наш PSP
                                ;
        mov     ax,3521h        ; Получаем адрес обработчика
        int     21h             ; прерывания 21H
        mov     word ptr old21s,es
        mov     word ptr old21o,bx
                                ;
        lea     dx,int21_entry  ;
        mov     ax,2521h        ; Устанавливаем вектор прерывания
        int     21h             ; 21H на новый обработчик

        pop     ds
        pop     es

        mov     dx,offset install
        int     27h

already:
        mov     al,ds:[082h]    ; Разбор командной строки
        cmp     al,'/'          ; и проверка на допустимый
        jne     bad_key         ; ключ для выгрузки программы
                                ; из памяти (/END или /end)
        mov     al,ds:[083h]
        cmp     al,'E'
        je      key_2
        cmp     al,'e'
        jne     bad_key
key_2:
        mov     al,ds:[084h]
        cmp     al,'N'
        je      key_3
        cmp     al,'n'
        jne     bad_key
key_3:
        mov     al,ds:[085h]
        cmp     al,'D'
        je      unload
        cmp     al,'d'
        jne     bad_key

unload:
        mov     ax,word ptr es:old21o ; Получение адресов для
        mov     dx,word ptr es:old21s ; восстановления вектора INT21H
        mov     cx,es           ; Сохраняем резидентную
                                ; CS часть программы
                                ;
        cli                     ;
        xor     bx,bx           ;
        mov     es,bx           ;
        mov     bx,21h*4        ;
        mov     es:[bx],ax      ; Восстановление оригинального
        mov     es:[bx+2],dx    ; вектора INT21H
        sti                     ;
                                ;
        mov     ah,49h          ;
        mov     es,cx           ; Восстановление ES
        int     21h             ; и освобождение памяти
                                ;
bad_key:
        mov     ax,4C00h        ; Завершение программы
        int     21h
code    ends
        end     start
```

■

Анонс

"Монитор" — версия 4.93

В статье рассказывается об основах организации гипертекстовых систем. Основное внимание уделено проблемам построения и применения гипертекстов. Предполагается опыт работы на C++. Первая часть посвящена вопросу создания гипертекстового транслятора.

Приведены интегральные характеристики отечественных инструментальных средств, представляющих разработки последних 5-7 лет. Рассмотрены конкретные отечественные инструментальные программные средства автоматизированной разработки прикладного программного обеспечения, представленные на проходивших в 1991-92гг. выставках Софтул, Информатика и других.

Вторая часть цикла статей о библиотеке Data Vision. Обсуждаются принципы построения видимых элементов в рамках Turbo Vision. Приведен исходный текст программы-редактора файлов баз данных Dbase и Paradox.

В статье обсуждается несколько различных подходов к разработке инструментальных средств, используемых при создании так называемых заказных прикладных систем. Сформулированы основные требования, которым должны удовлетворять эти средства. Представлено краткое описание комплекса инструментальных средств QuickTools, обеспечивающего проектирование меню прикладных систем, создаваемых на базе СУБД Paradox 3.5.

Оригинальный подход к реализации поддержки "мыши" в Clipper программах, а также клавиш-модификаторов Ctrl, Alt и Shift.

В статье рассказывается о новых возможностях, которые предоставляет программисту микропроцессор Pentium.

В статье рассматриваются основные прерывания обслуживания клавиатуры в среде операционной системы MS-DOS. Излагаются принципы создания демонстрационных вариантов программ, использующих в процессе своего выполнения ввод с клавиатуры компьютера.

Получение стереоскопических изображений на экране EGA монитора. Язык программирования: Turbo Pascal 6.0.

В статье предложен способ управления цветовой палитрой, использующий механизм изменения интенсивности основных цветов RGB палитры.

Подробно рассматриваются процедурные вопросы регистрации компьютерных программ в США.

Двенадцать примеров, разобрав которые, Вы убедитесь, что только начали свое знакомство с искусством программирования.

AVSP предоставляет пользователю возможность непосредственного ввода информации о новых вирусах в программу. Благодаря регулярной публикации такой информации AVSP на Вашем компьютере не отстанет от времени и создателей новых вирусов.

AVSP содержит ряд средств борьбы с неизвестными вирусами. ДИАЛОГ-МГУ обращается ко всем читателям с просьбой присылать информацию об обнаруженных новых вирусах.

Для ввода приведенной ниже информации в Вашу программу войдите в пункт меню "Данные о файлах и вирусах", подменю "Изменение информации о вирусах". Для более подробного описания можно обратиться к документации (пункт "Ввод информации о вирусах из журналов").

Имя вируса: SVC-1689.2
Номер: 236.1-24613
Тип вируса: PROGRAM
Размер: 1689
Шаблон: FE0000DA99EE032E8984
Процедуры: *.* { OPT(End:-1689) Mov(End:-80,Head:0,24) Tr(End:-1689) }
Справка: Штамм вируса SVC-1689. Заражает COM и EXE файлы. Резидентный.

Имя вируса: EGA-469
Номер: 237.1-64191
Тип вируса: PROGRAM
Размер: 469:485
Шаблон: 061EB800008EC00E1BE
Процедуры: *.EXE { Opt(End:-456) Mov(End:-466,Head:20,4) TrEXE(End:-469) }
Справка: На компьютере, зараженном этим вирусом, происходит плавное движение экрана вверх-вниз.

Имя вируса: New-Stone
Номер: 238.1-41077
Тип вируса: BOOT
Шаблон: D3E08EC0A3F07CB900020E1F33FF
Процедуры: #HardDisk { MovSect(0:0:6,0:0:1) }
Справка: Содержит текст "DAMCDOOM". На дискете лечится обнулением загрузочного кода.

Новая информация по вирусу Sum!

Имя вируса: Sum
Номер: 232.2-61299
Тип вируса: PROGRAM
Шаблон: 9C9D03079C4343E2F89D5BC3
Процедуры: *.COM { Opt(End:-849) Mov(End:-1218,Beg:0,30) Tr(End:-1231) }
Справка: Резидентный. Есть несколько мало отличающихся друг от друга версий.

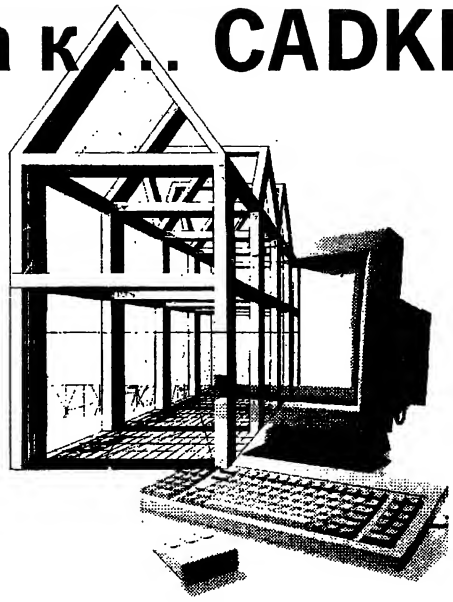
Имя вируса: AT-140
Номер: 239.1-8917
Тип вируса: PROGRAM
Размер: 140
Шаблон: 5E83C62CA4A5B835008E
Процедуры: *.COM { Opt(End:-134) Mov(MP:44,Beg:0,3) Tr(MP:-6) }
Справка: Вирус написан для побития рекорда длины на тело вируса. Не работает на компьютерах с процессором 8086/88.

По вопросам приобретения AVSP 2.0 обращайтесь в ДИАЛОГ-МГУ по телефонам (095) 939-38-90, 939-38-91. Подробная информация высылается бесплатно.

Цена одного комплекта AVSP 2.0 на 1 марта 1993 г. составляет 7200 рублей (с учетом 28% налога). При покупке нескольких комплектов предоставляется скидка. Скидка для дилеров — до 60%.

От кульмана к... CADKEY

В. Нефедов
Д. Самсонов



Введение в CADKEY

CADKEY, программный продукт фирмы CadKey Inc., является одной из лидирующих прикладных систем конструкторско-проектных работ для персональных компьютеров. CADKEY широко распространен за рубежом и является альтернативой хорошо известной российскому читателю графической системы AUTOCAD фирмы Autodesk.

CADKEY может широко использоваться в самых различных областях науки и техники:
– проектировании машин и механизмов;
– инженерном анализе;
– графическом отображении;
– трехмерном цифровом преобразовании;
– архитектурном проектировании;
– выпуске технической документации.

Вот далеко не полный перечень областей применения графической системы CADKEY.

Система имеет высокую производительность, которая достигается как собственными программными средствами, так и благодаря использованию CADL (языка программирования высокого уровня системы CADKEY), с помощью которого CADKEY может быть использован для решения практических задач проектирования. Благодаря трехмерным возможностям CADKEY редактирование проекций выполняется в один прием. Вместо того чтобы исправлять каждую проекцию Вашей конструкции, Вы модифицируете только одну, остальные изменяются автоматически.

Работа в среде CADKEY

После того как Вы приобрели графическую систему CADKEY, Вам необходимо установить ее на Ваш персональный компьютер. Для этого в пакет системы входит программа INSTALL.EXE, которая автоматически создает несколько рабочих поддиректорий и устанавливает в них систему.

Установив CADKEY, Вы должны определить конфигурацию CADKEY и его рабочие параметры программой CONFIG.EXE, входящей в пакет системы. Программа CONFIG работает с пользователем в режиме диалога и создает текстовый файл CONFIG.DAT, в котором сохраняются данные конфигурации системы после выхода из программы. CADKEY использует этот файл для настройки своих рабочих параметров после загрузки системы в оперативную память компьютера. При помощи программы CONFIG.EXE производится установка опций графического адаптера, устройств ввода, плоттера, принтера, настройка рабочих параметров программы базы данных системы, установка маршрута доступа к базе данных, используемого CADKEY по умолчанию, установка опций автоматической загрузки файлов.

После установки конфигурации CADKEY готов к работе. Для запуска системы выполните следующие действия:
– в строке приглашений DOS наберите команду CADKEY;
– нажмите клавишу Enter или Return.

После инициализации системы экран дисплея примет вид, изображенный на рис. 1, но с пустыми строкой приглашений и строкой маршру-

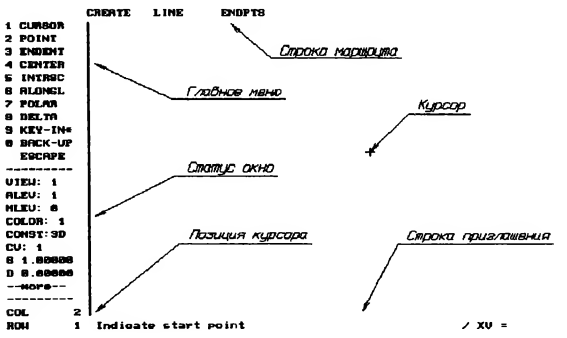


Рис. 1. Вид экрана при работе в среде CADKEY

та. Принцип работы в системе CADKEY заключается в перемещении по меню и выбору из него требуемых команд, составленных на английском языке. Текущий выбор отражается в строке маршрута системы.

В CADKEY существует два метода для выбора команд из любого меню. Для этого могут быть использованы как функциональные клавиши клавиатуры (F1...F10), так и планшет или "мышь". При этом клавиша Esc всегда используется для возвращения в главное меню из любых позиций, а функциональная клавиша F10 для отмены предыдущей выбранной команды. Некоторые часто используемые функции могут быть вызваны нажатием "горячих" клавиш, независимо от того, находитесь ли Вы в структуре меню. Например, одновременное нажатие клавиш Ctrl-I вызывает внутрисистемный калькулятор.

В статус-окне системы, которое представляет циклически изменяющийся массив окон, содержится информация о текущих рабочих параметрах чертежа. Здесь Вы можете изменять текущий вид дисплея, рабочие уровни, устанавливать цвет линий, их толщину, рабочий номер пера плоттера, атрибуты шрифта и размерных линий и другие опции CADKEY. Активизация статус-окна осуществляется выбором соответствующих команд в меню, курсором планшета или "мыши", или использованием "горячих" клавиш. Например, одновременное нажатие клавиш ALT-Y активизирует функцию изменения текущей ширины линии. При использовании курсора планшета или "мыши" циклическое перемещение по статус-окну осуществляется выбором команды MORE.

Строка приглашений, которая находится в самой нижней части экрана, снабжает Вас информацией и инструкциями на английском языке, помогающими Вам выбрать опции меню, ввести данные, выбрать элемент создаваемого Вами чертежа и разместить элемент в выбранной позиции.

Итак, наконец преодолев муки выбора, Вы среди огромного множества программных средств отдали предпочтение CADKEY и приобрели его. Что же дальше? Давайте попробуем с его помощью создать простенький чертеж прокладки, которая изображена на рис. 2.

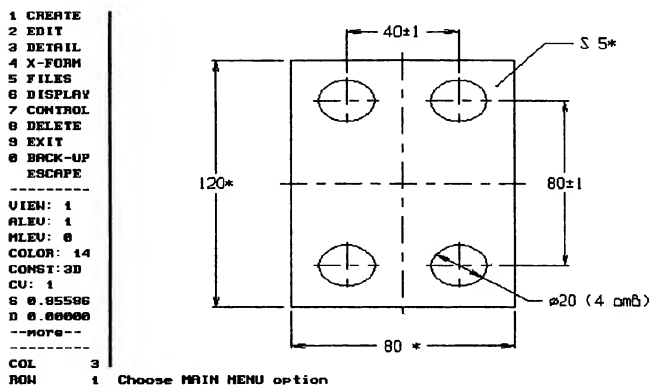


Рис. 2. Чертеж прокладки

Как создать чертеж в CADKEY?

Для создания чертежа воспользуемся набором типовых элементов, имеющихся в распоряжении системы CADKEY.

Будем считать, что в нашем распоряжении имеется плоттер с восьмипозиционной кареткой (например, Schlumberger или ему подобный). Тогда примем, что в первой позиции каретки будет находится тонкое перо, толщиной 0,25 мм, а во второй — толстое, толщиной 0,6мм. Поэтому в процессе вычерчивания элементов чертежа необходимо, в зависимости от требуемой толщины линии, переопределять позицию пера в каретке.

Изменить ее можно через главное меню с помощью следующей цепочки команд:

CONTROL(контроль)–ATTRIB(атрибуты)–PEN#(номер позиции пера)–CURRENT(текущее).

Enter new pen number from 1-8 (current=1) => 1 или 2 <CR>
Введите номер пера от 1 до 8 (текущее =1) => 1 или 2 <CR>

Установить номер пера можно также в статус-окне в пункте "PEN:".

Если используется одноперьевой плоттер, то для вычерчивания толстых линий необходимо задавать плоттеру многопроходный режим работы, то есть менять ширину элементов чертежа.

Ширина элементов устанавливается через главное меню с помощью следующей цепочки команд:

CONTROL(контроль)–ATTRIB(атрибуты)–
L-WIDTH(ширина)–CURRENT(текущее).

Enter new line width as odd number from 1-15 (current=1) => 1 или 3 <CR>
Введите непарное значение толщины линии от 1 до 15(текущее =1)=> 1 или 3 <CR>

Установить толщину линии можно также в статус-окне в пункте "L-WID:".

Изображение прокладки начнем с прорисовывания внешнего контура с помощью примитива POLYLIN (полилиния).

Выберем в главном меню следующую цепочку команд:

CREATE(создать)–POLYLIN(полилиния)–RECTANG (прямоугольник)–WID/HT(по высоте и ширине).

Enter width of rectangle (10.00) => 80 <CR>
Введите ширину прямоугольника (10.00) => 80 <CR>

Enter height of rectangle (10.00) => 120 <CR>
Введите высоту прямоугольника (10.00) => 120 <CR>

Indicate lower left corner
Укажите левый нижний угол

При установке левого нижнего угла из меню точки выберем следующую команду:

KEY-IN (ввод координаты точки с клавиатуры).

Indicate point/ XV = -40 <CR>
Укажите точку YV = -60 <CR>
ZV = 0 <CR>

Для отображения всего чертежа в текущем окне нажмите комбинацию клавиш Alt-A (автоматическое масштабирование).

На экране с центром в точке с координатами (0,0,0) появится прямоугольник шириной 80 мм и высотой 120 мм.

Для выхода из повтора текущей команды в главное меню нажмите на клавишу ESC.

Изобразим левое нижнее отверстие. Для построения отверстия воспользуемся в главном меню следующей цепочкой команд:

CREATE(создать)–CIRCLE(окружность)–CTR+DIA(по центру и диаметру)

Enter diameter (10.00) = 20 <CR>
Введите диаметр (10.00) = 20 <CR>

При задании центра окружности из меню точки выберем следующую команду:

KEY-IN (ввод координаты точки с клавиатуры).

Indicate point / XV = -20 <CR>
Укажите точку YV = -40 <CR>
ZV = 0 <CR>

На экране с центром в точке с координатами (-20, -40, 0) появится окружность с диаметром, равным 20 мм.

Для выхода из повтора текущей команды в главное меню нажмите на клавишу ESC.

Построим осевые линии.

Перед построением осевых линий необходимо изменить тип линии (сплошную на штрих-пунктирную), воспользовавшись в статус-окне пунктом "L-TYPE:".

Для построения осевых линий выберем следующую цепочку команд:

CREATE(создать)–LINE(линия)–HRZ/PRP(горизонтальная /перпендикулярная)–BOTH(пересекающиеся)–KEY-IN (ввод центра с клавиатуры).

Indicate point / XV = 0 <CR>
Укажите точку YV = 0 <CR>
ZV = 0 <CR>

На экране появятся осевые линии, пересекающиеся в точке с координатами (0,0,0), ограниченные по длине размерами текущего окна.

Если осевые линии выходят за размеры контура на слишком большую величину, то их желательно подрезать. Для этого можно провести на необходимом расстоянии от контура прямоугольника отрезки, пересекающие осевые линии и, воспользовавшись соответствующими командами из меню EDIT (редактировать), подрезать линии. После этого ненужные линии необходимо стереть, воспользовавшись комбинацией клавиш Ctrl-Q.

Точно так же изображаются осевые линии отверстия, с той лишь разницей, что для установки центра прохождения осей из меню точки выбирается опция CENTER (по центру элемента).

Изобразим оставшиеся три отверстия, воспользовавшись зеркальным отражением имеющегося отверстия относительно вертикальной оси, а полученные два отверстия — относительно горизонтальной оси.

Для выполнения вертикального отражения выберем из главного меню следующую цепочку команд:

X-FORM (изменение формы)–MIRROR (отражение)–COPY (с сохранением отражаемого объекта)–WINDOW (выбор отражаемых элементов в окно).

Indicate position for 1st selection window corner
Укажите позицию первого угла выбранного окна

Indicate position for 2nd selection window corner
Укажите позицию второго угла выбранного окна

Indicate 1st position for mirror
Укажите первую позицию для отражения

Indicate 2nd position for mirror
Укажите вторую позицию для отражения

Произойдет отражение окружности с осевыми линиями относительно вертикальной оси.

Для отражения имеющихся двух отверстий относительно горизонтальной оси необходимо вернуться на шаг назад (для этого необходимо нажать на F10), повторить выбор элементов для отражения (с учетом уже отраженной окружности) и указать в качестве позиций для отражения левую крайнюю и правую крайнюю точки горизонтальной оси.

Для выхода из повтора текущей команды в главное меню нажмите на клавишу ESC.

Перед построением размерных линий с указанием размеров необходимо установить параметры этих линий, которые не устанавливаются автоматически при загрузке CADKEY. К этим параметрам относятся:
– вид маркера (стрелка или точка);
– направление стрелки (внутреннее или внешнее);
– возможность редактирования величины размера.

Вид маркера устанавливается через главное меню по следующей цепочке команд:

DETAIL(оформление чертежа)–SET(параметры)–ARROWS(стрелки)- STYLE(вид)–ARROW(стрелка).

Направление стрелки задается в статус-окне с помощью пункта "ARR:".

Для установки возможности редактирования величины размера воспользуемся следующей цепочкой команд из главного меню:

DETAIL(оформление чертежа)– SET(параметры)–STANDRD(образец)–DIM MOD(режим)–MANUAL(ручное редактирование).

Проставим горизонтальное межцентровое расстояние. Для этого используем из главного меню следующую цепочку команд:

DETAIL(оформление чертежа)–DIMENSN(размеры)–HORIZTL(горизонтально).

Indicate 1st position
Укажите начало первой выносной линии

Indicate 2nd position
Укажите начало второй выносной линии

Enter dimension text string (40.00): 40 Alt241 1 <F6>
Введите текстовую строку размером (40.00): 40 Alt241 1 <F6>

Indicate dimension text position
Укажите позицию текста

После простановки размера в строке приглашений появляется сообщение:

Accept dimension? (CR for YES)
Принять размер? (CR, если Да)

Если проставленный размер удовлетворяет Вас, то нажмите <CR>, а если нет, выберите в меню "NO" и укажите новую позицию текста.

Проставим вертикальное межцентровое расстояние. Для этого используем из главного меню следующую цепочку команд:

DETAIL(оформление
чертежа)–DIMENSN(размеры)–VERTICL(вертикально).

Indicate 1st position
Укажите начало первой выносной линии

Indicate 2nd position
Укажите начало второй выносной линии

Enter dimension text string (80.00): 80 Alt241 1 <F6>
Введите текстовую строку размером (80.00): 80 Alt241 1 <F6>

Indicate dimension text position
Укажите позицию текста

После простановки размера в строке приглашений появляется сообщение:

Accept dimension? (CR for YES)
Принять размер? (CR, если Да)

Если проставленный размер удовлетворяет Вас, то нажмите <CR>, а если нет, выберите в меню "NO" и укажите новую позицию текста.

Проставим горизонтальный габаритный размер. Для этого используем из главного меню следующую цепочку команд:

DETAIL(оформление
чертежа)–DIMENSN(размеры)–HORIZTL(горизонтально).

Indicate 1st position
Укажите начало первой выносной линии

Indicate 2nd position
Укажите начало второй выносной линии

Enter dimension text string (80.00): 80* <F6>
Введите текстовую строку размером (80.00): 80* <F6>

Indicate dimension text position
Укажите позицию текста

После простановки размера в строке приглашений появляется сообщение:

Accept dimension? (CR for YES)
Принять размер? (CR, если Да)

Если проставленный размер удовлетворяет Вас, то нажмите <CR>, а если нет, выберите в меню "NO" и укажите новую позицию текста.

Проставим вертикальный габаритный размер. Для этого используем из главного меню следующую цепочку команд:

DETAIL(оформление
чертежа)–DIMENSN(размеры)–VERTICL(вертикально).

Indicate 1st position
Укажите начало первой выносной линии

Indicate 2nd position
Укажите начало второй выносной линии

Enter dimension text string (120.00): 120* <F6>
Введите текстовую строку размером (120.00): 120* <F6>

Indicate dimension text position
Укажите позицию текста

После простановки размера в строке приглашений появляется сообщение:

Accept dimension? (CR for YES)
Принять размер? (CR, если Да)

Если проставленный размер удовлетворяет Вас, то нажмите <CR>, а если нет, выберите в меню "NO" и укажите новую позицию текста.

Проставим диаметр отверстия. Для этого используем из главного меню следующую цепочку команд:

DETAIL(оформление чертежа)– DIMENSN(размеры)–
DIAMETR(диаметр).

Select arc to be dimensioned
Выберите окружность для простановки размера

Enter dimension text string (20.00): Alt252 20 (4 отв.) <F6>
Введите текстовую строку размером (20.00): Alt252 20 (4 отв.) <F6>

Indicate dimension text position
Укажите позицию текста

Choose leader side (current=LEFT) <CR>
Введите сторону простановки полки (текущая-левая) <CR>

После простановки размера в строке приглашений появляется сообщение:

Accept dimension? (CR for YES)
Принять размер? (CR, если Да)

Если проставленный размер удовлетворяет Вас, то нажмите <CR>, а если нет, выберите в меню "NO" и укажите новую позицию текста.

Проставим ширину прокладки. Ширину прокладки проставим примитивом LABEL (выноска). Перед использованием его необходимо поменять вид маркера, который устанавливается через главное меню по следующей цепочке команд:

DETAIL(оформление
чертежа)–SET(параметры)–ARROWS(стрелки)-
STYLE(вид)–POINT(точка).

Для простановки выноски воспользуемся из главного меню следующей цепочкой команд:

DETAIL(оформление чертежа)–LABEL(выноска).

Enter text): S 5* <F6>
Введите текст: S 5* <F6>

Indicate text position
Укажите позицию текста

Choose leader side (current=LEFT) <CR>
Введите сторону простановки полки (текущая-левая)
<CR>

Indicate arrow tip position
Укажите конечную позицию маркера

После простановки выноски в строке приглашений появляется сообщение:

Accept label? (CR for YES)
Принять выноску? (CR, если Да)

Если проставленная выноска удовлетворяет Вас, то нажмите <CR>, а если нет, выберите в меню "NO" и укажите новую позицию маркера.

На этом создание чертежа прокладки можно считать законченным. Теперь нам осталось записать полученный чертеж на диск, для чего выполним следующую цепочку команд:

FILES(работа с файлами)–PART(файл с расширением
PRT)–SAVE(сохранить).

Enter part file name(): PROKLAD <CR>
Введите имя файла(): PROKLAD <CR>.

После нажатия клавиши <CR> чертеж будет записан в файл с именем proklad.prt в поддиректории CADKEY PRT.

Как повысить эффективность использования CADKEY

Наконец Вы получили чертеж прокладки на экране дисплея и сохранили его на диске. Вам, наверное, покажется очень долгим процесс создания такого простого чертежа, хотя, если за компьютером работает опытный оператор, то время, которое он затратит на создание такой прокладки, будет не больше того времени, которое потратите Вы при создании такого чертежа за кульманом. Однако наибольший эффект от использования CADKEY Вы получите, если Ваш чертеж имеет много однотипных деталей. В этом случае у Вас есть три варианта создания копий прокладки.

Первый вариант — это использование аффинных преобразований и создание копий прокладки командой X-FORM (изменение формы), которая предоставляет возможности перемещения по осям, выполнения поворота относительно выбранной точки, изменения размера, путем ввода масштабного коэффициента, выполнения операции зеркального отражения.

Второй, более гибкий, — это создание файла типа pattern (образец). Для создания образца прокладки выберем из главного меню следующую цепочку команд:

FILES (файлы)–PATTERN
(образец)–CREATE(создать)–WINDOW (окно).

Курсором укажем окно, в котором должны находиться все элементы прокладки. В окне приглашений появится запрос на ввод базовой точки:

Indicate pattern base position
Укажите базовую позицию

В окне главного меню появится меню выбора позиций. Выберем базовую точку вводом с клавиатуры команды

KEY-IN

в меню выбора позиций и введем нулевые координаты по всем трем осям в ответ на запрос CADKEY. После этого Вы должны ввести имя файла, например, proklad в ответ на запрос системы

Enter pattern file name ()
Введите имя файла образцов

и нажать клавишу <CR>. CADKEY сохранит образец Вашей прокладки в файле с именем proklad.ptn в директории PTN. Теперь Вы всегда можете воспользоваться образцом прокладки и загрузить ее в любое место Вашего чертежа. Для этого выполните следующие команды:

FILES (файлы)–PATTERN (образец)–RETRIEV
(воссоздать).

В строке приглашений появится запрос

Enter pattern file name ()
Введите имя файла образца

В ответ наберите имя файла, в котором Вы сохранили образец Вашей прокладки, например, proklad и нажмите <CR>. В окне меню выберите команду

NO (нет)

не создавать группу. Если Вы выберете команду

YES (да),

то все элементы прокладки будут объединены в группу, то есть прокладка будет представлять единый элемент Вашего чертежа. В следующем появившемся меню выберите команду

CUR-LEV (вывод на текущий уровень)

и дайте ответы на последующие запросы:

Enter scale factor (1.000)
Введите масштабный коэффициент (1.000);

```
Enter rotation angle (0.000)
Введите угол поворота (0.000),
```

после этого из меню выбора позиции выберите знакомую Вам команду

```
KEY-IN (ввод с клавиатуры)
```

и в строке приглашений введите координаты

```
/XV= 500
/YV = 0
/ZV = 0.
```

В точке с указанными координатами появится копия Вашей прокладки. Одновременным нажатием клавиш Alt-A промасштабируйте Ваш чертеж, и Вы увидите созданную Вами копию прокладки. Однако данный способ не позволяет Вам изменять горизонтальные или вертикальные размеры прокладки, расстояния между отверстиями, размеры отверстий. Но CADKEY поможет Вам преодолеть эти неудобства. Третий вариант создания копий прокладки — это использование программирования на языке высокого уровня CADL.

Программирование на CADL

При помощи языка высокого уровня CADL Вы можете создать параметрическую прокладку, где Вы можете по своему усмотрению задавать размеры прокладки, и все это будет отображаться на мониторе.

Язык CADL по своему синтаксису близок языку BASIC, поэтому обучение программированию на нем и работа не составляют особого труда.

Итак, наша программа (листинг 1) должна обеспечить ввод информации обо всех размерах прокладки, рисовать прокладку как без размерных линий (для сборочных чертежей), так и с размерами (для деталировок). Также в программе должно быть предусмотрено изменение точности допуска на линейные размеры и размеры диаметров отверстий.

Текст программы Вы можете набрать в любом текстовом редакторе. Файл с текстом программы Вы должны поместить в поддиректорию \CDL, которая используется для файлов с расширением CDL. Присвоим файлу имя proklad.cdl. Вы можете в CADKEY работать как с откомпилированными программами (для этого в Вашем распоряжении имеется компилятор языка CCOMP, который создает двоичный код программы, что увеличивает скорость выполнения Вашего приложения), так и с текстовыми файлами. Для простоты рассмотрим вариант выполнения нашей неоткомпилированной программы. В среде CADKEY для запуска программы из главного меню выберем цепочку команд:

```
FILES (файлы)–CADL (CDL программы)–EXECUTE
(выполнить).
```

В строке приглашений появится надпись

```
Enter CADL filename ()
Введите имя CADL файла
```

В ответ наберите имя файла proklad и нажмите <CR>. После загрузки программы в окне главного меню системы появится меню нашей программы, содержащее следующие команды:

```
1. Assembly;
2. Detail;
3. Setup;
4. Exit.
```

По командам ASSEMBLY и DETAIL выполняется создание чертежа прокладки без показа размерных линий и с выводом размерных линий на монитор. После запуска этих команд Вы должны последовательно в строке приглашений ввести размеры, необходимые для создания прокладки, при этом в скобках появляются размеры, которые Вы можете использовать по умолчанию, для чего просто нажимайте <CR>, или вводите свои значения с клавиатуры, ввод подтверждайте нажатием клавиши <CR>. Вы должны задать следующие параметры прокладки:

```
Enter width rectangle (80) =>
Введите ширину прямоугольника;

Enter hight rectangle (120) =>
Введите высоту прямоугольника;

Enter diameter (20) =>;
Введите диаметр;

Enter distance between circle on horizontal (40) =>
Введите расстояние между отверстиями по
горизонтали;

Enter distance between circle on vertical (80) =>
Введите расстояние между отверстиями по вертикали;

Enter width (1.0) =>
Введите толщину прокладки.
```

После этого появится меню выбора позиции и программа запросит у Вас координаты точки на экране, в которую она затем поместит левый нижний угол прокладки и выведет ее на экран в заданном месте. По команде SETUP Вы можете установить допуски на линейные размеры и размер отверстия в прокладке. При выборе команды EXIT происходит возврат в главное меню CADKEY.

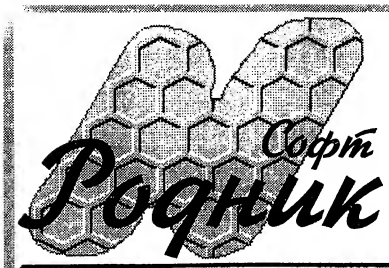
Таким образом, Вы можете ориентировать CADKEY на решение Ваших прикладных задач, что значительно повысит эффективность использования CADKEY и сократит время проектирования Вашего изделия. Недостатком данной версии CADKEY (версия 3.0) является необходимость создания меню и текста на английском языке в строке приглашений в Ваших приложениях на CADL. В более поздних версиях это маленькое неудобство устранено.

Заключение

Надеемся, что данная статья принесла Вам хоть немного пользы и Вы не зря потратили время на ее чтение. В меру своих сил мы постарались кратко ознакомить Вас, уважаемый читатель, с принципами работы в CADKEY и показали самые скромные возможности, которыми располагает данная графическая система. ■

Листинг 1

```
rem ********* Инициализация переменных ******************
      menopt    = 1
      DX        = 80
      DCX       = 40
      DY        = 120
      DCY       = 80
      D         = 20
      S         = 1
      Col       = @color
      LTol      = 1
      DTol      = 0.1
:mainmenu     rem Создать главное меню
      getmenu "","Assemble","Detail","Setup","Exit"
            on @key goto mainmenu, getdem, getdemc, setup, quit
rem ********* Выводить размерные линии? ********************
:getdem       rem Нет
      dim = 1
      goto enterbasedim
:getdemc      rem Да
      dim = 0
:enterbasedim       rem Задать базовые размеры
      getflt "Enter width (%f) = > ", DX, DX
      getflt "Enter heigth (%f) = > ",DY, DY
      getflt "Enter diameter (%f) = > ",D, D
      getflt "Enter distance between circle on horizontal (%f) = > ", DCX, DCX
      getflt "Enter distance between circle on vertical (%f) = > ", DCY, DCY
      getflt "Enter width (%f) = > ", S, S
:getbasepoint       rem Указать базовую точку вывода на экран
      GETPOS "Indicate base point of position", menopt
      rem Если точка не задана, в зависимости от нажатой клавиши
      rem сделать переход
            on ( @key + 4 ) goto getbasepoint, mainmenu, enterbasedim, getcoord
:getcoord
      rem Получить координаты базовой точки из системных переменных
      Xb = @xworld
      Yb = @yworld
      Zb = @zworld
      rem Запомнить текущую позицию в меню GETPOS
      menopt = @key
      goto evalcoord
:evalcoord          rem Определить координаты базовых точек
      rem ********** Базовые точки прямоугольника **************
      rem ***** Первая базовая точка Xb и Yb левого нижнего угла ******
      X2 = Xb+DX
      Y2 = Yb+DY
      rem ********* Базовые точки отверстий ***********************
      CX1 = Xb+(DX-DCX)/2
      CY1 = Yb+(DY-DCY)/2
      CX2 = CX1+DCX
      CY2 = CY1+DCY
:drawpr
      rem ******** Рисовать прямоугольник *********************
      LINE Xb, Yb, Zb, X2, Yb, Zb, Col, , 1, 0, 0, 2, 1
      LINE X2, Yb, Zb, X2, Y2, Zb, Col, , 1, 0, 0, 2, 1
      LINE X2, Y2, Zb, Xb, Y2, Zb, Col, , 1, 0, 0, 2, 1
      LINE Xb, Y2, Zb, Xb, Yb, Zb, Col, , 1, 0, 0, 2, 1
      rem ********** Рисовать осевые линии прямоугольника *********
      LINE Xb+DX/2, Yb-1, Zb, Xb+DX/2, Y2+1, Zb, 14, , 3, 0, 0, 1, 1
      LINE Xb-1, Yb+DY/2, Zb, X2+1, Yb+DY/2, Zb, 14, , 3, 0, 0, 1, 1
      rem ***** Рисовать окружности и их осевые линии *************
      CIRCLE  CX1, CY1, Zb, D/2, 0, Col, , 1, 0, 0, 2, 1
      LINE    CX1-D/2-1, CY1, Zb, CX1+D/2+1, CY1, Zb, 14, , 3, 0, 0, 1, 1
      LINE    CX1, CY1-D/2-1, Zb, CX1, CY1+D/2+1, Zb, 14, , 3, 0, 0, 1, 1
      CIRCLE  CX2, CY1, Zb, D/2, 0, Col, , 1, 0, 0, 2, 1
      LINE    CX2-D/2-1, CY1, Zb, CX2+D/2+1, CY1, Zb, 14, , 3, 0, 0, 1, 1
      LINE    CX2, CY1-D/2-1, Zb, CX2, CY1+D/2+1, Zb, 14, , 3, 0, 0, 1, 1
      CIRCLE  CX2, CY2, Zb, D/2, 0, Col, 1, 1, 0, 0, 2, 1
      LINE    CX2-D/2-1, CY2, Zb, CX2+D/2+1, CY2, Zb, 14, , 3, 0, 0, 1, 1
      LINE    CX2, CY2-D/2-1, Zb, CX2, CY2+D/2+1, Zb, 14, , 3, 0, 0, 1, 1
      CIRCLE  CX1, CY2, Zb, D/2, 0, Col, 1, 1, 0, 0, 2, 1
      LINE    CX1-D/2-1, CY2, Zb, CX1+D/2+1, CY2, Zb, 14, , 3, 0, 0, 1, 1
      LINE    CX1, CY2-D/2-1, Zb, CX1, CY2+D/2+1, Zb, 14, , 3, 0, 0, 1, 1
      if ( dim = = 1 )
            goto mainmenu
rem ********** Рисовать размерные линии ******************
rem создадим массив с координатами базовых точек
ARRAY   GENDAT[18]
            GENDAT[0]   = Xb-21
            GENDAT[1]   = Yb
            GENDAT[2]   = Xb
            GENDAT[3]   = Yb
            GENDAT[4]   = Xb-21
            GENDAT[5]   = Y2
            GENDAT[6]   = Xb
            GENDAT[7]   = Y2
            GENDAT[8]   = Xb-20
            GENDAT[9]   = Yb
            GENDAT[10]  = Xb-20
            GENDAT[11]  = Y2
            GENDAT[12]  = Xb-20
            GENDAT[13]  = Yb
            GENDAT[14]  = 90
            GENDAT[15]  = Xb-20
            GENDAT[16]  = Y2
            GENDAT[17]  = 270
      rem прочитать в $text значение DY и LTol
      SPRINT $text, "%.1f ё %.2f", DY, LTol
      rem определить длину строки $text
      CALL strlen, $text, lentxt
      rem определить длину строки $text в миллиметрах
      len = lentxt*@txtasp*@noteht
      rem вывести размер
      GENDIM 50, 3, 0, 2, GENDAT, 2, Xb-22 , Yb+DY/2-len/2, $text,\
            90, , , 0, 14, 0, 0, 0, 0, 1
            GENDAT[0]   = CX1-D/2-1
            GENDAT[1]   = CY1
            GENDAT[2]   = Xb-11
            GENDAT[3]   = CY1
            GENDAT[4]   = CX1-D/2-1
            GENDAT[5]   = CY2
            GENDAT[6]   = Xb-11
            GENDAT[7]   = CY2
            GENDAT[8]   = Xb-10
            GENDAT[9]   = CY1
            GENDAT[10]  = Xb-10
            GENDAT[11]  = CY2
            GENDAT[12]  = Xb-10
            GENDAT[13]  = CY1
            GENDAT[14]  = 90
            GENDAT[15]  = Xb-10
            GENDAT[16]  = CY2
            GENDAT[17]  = 270
      SPRINT $text, "%.1f ё %.2f", DCY, LTol
      CALL strlen, $text, lentxt
      len = lentxt*@txtasp*@noteht
      GENDIM 50, 3, 0, 2, GENDAT, 2, Xb-12 , Yb+DY/2-len/2, $text,\
            90, , , 0, 14, 0, 0, 0, 0, 1
            GENDAT[0]   = Xb
            GENDAT[1]   = Yb
            GENDAT[2]   = Xb
            GENDAT[3]   = Yb-21
            GENDAT[4]   = X2
            GENDAT[5]   = Yb
            GENDAT[6]   = X2
            GENDAT[7]   = Yb-21
            GENDAT[8]   = Xb
            GENDAT[9]   = Yb-20
            GENDAT[10]  = X2
            GENDAT[11]  = Yb-20
            GENDAT[12]  = Xb
            GENDAT[13]  = Yb-20
            GENDAT[14]  = 0
            GENDAT[15]  = X2
            GENDAT[16]  = Yb-20
            GENDAT[17]  = 180
      SPRINT $text, "%.1f ё %.2f", DX, LTol
      CALL strlen, $text, lentxt
      len = lentxt*@txtasp*@noteht
      GENDIM 50, 3, 0, 2, GENDAT, 2, Xb+DX/2-len/2 , Yb-19, $text,\
            0, , , 0, 14, 0, 0, 0, 0, 1
            GENDAT[0]   = CX1
            GENDAT[1]   = CY1-D/2-1
            GENDAT[2]   = CX1
            GENDAT[3]   = Yb-11
            GENDAT[4]   = CX2
            GENDAT[5]   = CY1-D/2-1
            GENDAT[6]   = CX2
            GENDAT[7]   = Yb-11
            GENDAT[8]   = CX1
            GENDAT[9]   = Yb-10
            GENDAT[10]  = CX2
            GENDAT[11]  = Yb-10
            GENDAT[12]  = CX1
            GENDAT[13]  = Yb-10
            GENDAT[14]  = 0
            GENDAT[15]  = CX2
            GENDAT[16]  = Yb-10
            GENDAT[17]  = 180
      SPRINT $text, "%.1f ё %.2f", DCX, LTol
      CALL strlen, $text, lentxt
      len = lentxt*@txtasp*@noteht
      GENDIM 50, 3, 0, 2, GENDAT, 2, Xb+DX/2-len/2 , Yb-9, $text,\
            0, , , 0, 14, 0, 0, 0, 0, 1
      Ang = atan2(CY1-Yb, X2+10-CX2)
      SPRINT $text, "%.1f ё %.2f", D, DTol
      CALL strlen, $text, lentxt
      len = lentxt*@txtasp*@noteht
            GENDAT[0]   = X2+10
            GENDAT[1]   = Yb
            GENDAT[2]   = CX2-(D/2+8)*cos(Ang)
            GENDAT[3]   = CY1+(D/2+8)*sin(Ang)
            GENDAT[4]   = X2+10
            GENDAT[5]   = Yb
            GENDAT[6]   = X2+15+len
            GENDAT[7]   = Yb
            GENDAT[8]   = CX2-D/2*cos(Ang)
            GENDAT[9]   = CY1+D/2*sin(Ang)
            GENDAT[10]  = 180-Ang
            GENDAT[11]  = CX2+D/2*cos(Ang)
            GENDAT[12]  = CY1-D/2*sin(Ang)
            GENDAT[13]  = 360-Ang
      GENDIM 50, 2, 0, 2, GENDAT, 2, X2+12 , Yb+1, $text,\
            0, , , 0, 14, 0, 0, 0, 0, 1
      SPRINT $text, "S%.1f", S
      CALL strlen, $text, lentxt
      len = lentxt*@txtasp*@noteht
      LABEL   X2+15+len, Y2, X2+10, Y2, X2-10, CY2-DY/4, 1,\
            X2+12 , Y2+1, $text, 0, , , 0, 14, 0, 0, 0, 0, 1
      goto mainmenu
rem****** Установить величину допуска **********************
:setup
      getflt "Enter tolerance line (%f) = > ", LTol, LTol
      getflt "Enter tolerance diametr (%f) = > ",DTol, DTol
      goto mainmenu
rem************ Возврат в CADKEY **************************
:quit
      exit
```



113556 Москва, Нахимовский проспект 1, корп. 1.
Телефон: (095) 113-2688
Факс: (095) 316-9754

официальный представитель трех
американских компьютерных фирм

'Начинает и Выигрывает'!

Intersys
/USA/
Персональные компьютеры.

Никаких преград!
- **Индустриальные** ПК работают в экстремальных условиях: при вибрации, перегрузках и температуре до +60 градусов.
- **ПК с повышенной отказоустойчивостью** сделают Ваш сервер неуязвимым.
- **Промышленные SPARC** станции

ARES
Microdevelopment
/USA/
Персональные компьютеры.

На века!
- сверхжесткие требования к качеству комплектующих.
- многочасовое тестирование **каждой системы** обеспечивают небывалую долговечность.

ПЯТИЛЕТНЯЯ *гарантия!!!*

Texas Micro
/USA/
Индустриальные компьютеры.

Новейшие технологии!
- шина LOCAL BUS (производительность до 10 раз выше, чем у шины EISA).
- процессоры 486DX2.
- графические акселераторы.
- до 64 Мб оперативной памяти.

ДВУХЛЕТНЯЯ *гарантия!!!*

'Играющим' вместе с нами обеспечен успех!

Итак, Господа, ход за Вами ...

ELITSoft

Пользователям вычислительной техники Акционерное общество "ЭЛИТ СОФТ" предлагает

Агрегаты Бесперебойного Питания
(Uninterruptable Power Source)

Основные характеристики
- Мощность 900 ВА
- Время автономной работы 40 мин.
- Время переключения 5-7 мсек.
- Рабочий диапазон 180-240 В
- Гарантия 1 год
- Послегарантийное обслуживание

В конструкции предусмотрены
- Фильтрация сетевых помех
- Звуковая и световая сигнализация
- Индикация перегрузки
- Индикация уровня заряда батарей
- Герметичные, необслуживаемые аккумуляторные батареи

Поставка и оплата

Поставка со складов в Москве, Санкт-Петербурге, Чите, Рязани.
Любая удобная Вам форма оплаты.

Адрес: 105215 Москва, 11-я Парковая, 44, корп. 1

Телефон:
секретариат: (095) 965-35-29, 532-96-14
отдел сбыта: (095) 965-35-28, 965-35-30
Факс (095) 965-35-28, 965-35-30
Телетайп 205211 КТУР

Мы продаем компьютеры!

Юридическая защита программ в России: практическая реализация законодательства

М. Корсаков

Принятый осенью 1992 года Закон РФ "О правовой охране программ для электронно-вычислительных машин и баз данных" заложил солидную юридическую основу для защиты интересов разработчиков ПО с помощью института авторского права. Он представляет собой первый в истории России специальный законодательный акт, относящийся к программному обеспечению и охватывающий, насколько это возможно в рамках авторского права, весь спектр вопросов создания и использования программ и баз данных.

Даже краткое знакомство с текстом закона говорит о том, что при его создании максимально учитывался мировой опыт защиты ПО. Это нашло отражение в статьях Закона, устанавливающих следующие основные принципы регулирования отношений в области ПО.

Что охраняется Законом

К охраняемым объектам принадлежат все виды программ на любом языке, включая аудиовизуальные отображения, а также базы данных. Идеи, принципы построения программы и языки программирования к таким объектам не относятся. Данное обстоятельство снижает степень надежности защиты программного продукта и вызывает необходимость комплексной его охраны. Авторское право на программу возникает в силу ее создания и не требует осуществления каких-либо формальностей. Срок действия авторского права устанавливается в течение всей жизни автора и пятидесяти лет после его смерти.

Кто признается автором

Автором является физическое лицо, в результате творческой деятельности которого создана программа или база данных. В случае совместной разработки программы автором ее признается каждый из работавших. Автору программы принадлежат следующие личные права: право считаться автором программы или базы данных, право на имя и на неприкосновенность программы. Помимо личных, автор обладает рядом исключительных имущественных прав: на выпуск программы или базы данных в свет, их воспроизведение, модификацию и распространение, а также на любое иное использование своего продукта.

Передача прав

Автор может передать свои права третьим лицам на основе договора, определяющего объем и способы использования этими лицами программ для ЭВМ и баз данных, размер и порядок выплаты вознаграждения и срок действия договора. Таким образом, владелец прав на программу или

базу не обязательно является автором. Примером такой ситуации являются отношения разработчика программы и работодателя. Согласно закону, все имущественные права на программу или базу данных, созданную в рамках выполнения служебных обязанностей или по заданию работодателя, принадлежат последнему. Однако в договоре с работодателем может определяться иной порядок распределения имущественных прав на программу или базу данных.

По закону автор имеет право на **регистрацию программы** в Российском агентстве по правовой охране программ для ЭВМ, баз данных и топологий интегральных микросхем (РосАПО). Такая регистрация призвана облегчить процесс идентификации программы и установления авторства при различных спорах по поводу авторских прав.

Использование программы третьими лицами осуществляется за вознаграждение по договору с владельцем авторских прав на ПО. В законе предусмотрены случаи свободного использования программы законным пользователем; основные из них следующие:

а) любые действия, связанные с нормальным функционированием программы, например, запись ее на жесткий диск или исправление явных ошибок;

б) адаптация программы или базы данных;

в) изготовление архивной копии;

г) декомпилирование (в том случае, если это необходимо для обеспечения взаимодействия программ и информация для этого не может быть получена иными способами);

д) свободная перепродажа законно приобретенного экземпляра программы или базы данных.

В случае нарушения авторских прав копии программы, изготовление или использование которых ведет к таким нарушениям, считаются контрафактными и могут быть конфискованы. Интересно, что закон блокировал возможность нелегального ввоза копий охраняемой в России программы или базы данных из стран, где эти продукты никогда не охранялись или где их охрана прекратилась. Такие копии также признаются контрафактными, а их ввоз — нарушением авторского права.

В случае нарушения своих прав автор или правообладатель имеет право обратиться с иском в суд или арбитраж (третейский суд) с требованием восстановления нарушенных прав и выплаты по выбору либо суммы убытков, либо компенсации за упущенную выгоду в размере от пяти до пятидесяти тысяч минимальных заработных плат. Нарушение же личных прав автора, в частности, выпуск под своим именем чужой программы или базы данных или их незаконное воспроизведение и распространение влечет за собой уголовную ответственность.

Необходимо отметить, что в полной мере механизм защиты прав на программное обеспечение будет действовать в лучшем случае с 1994 года в силу объективных причин, связанных с приведением в соответствие закону большого числа нормативных актов и созданием механизма его практической реализации. Это, кстати, отражено и в постановлении о введении закона в действие.

Однако уже сейчас при практической реализации законодательных норм возникают следующие важные моменты.

Официальная регистрация

Являясь необязательной, она, тем не менее, значительно облегчает рассмотрение споров о нарушении прав, а также повышает статус разработки при отношениях с потребителями. Кроме того, сведения о зарегистрированной программе публикуются в официальных изданиях Агентства. Процедура регистрации состоит в оформлении заявки и направлении ее, описания программы (реферата) и части исходного текста в Агентство по адресу:

103621 Москва, Малый Черкасский пер., д. 2/6.
Телефон: 206-10-82.

В заявке приводятся полное и сокращенное названия программы, предыдущее или альтернативное название, сведения о публикациях, дата окончательного завершения работы, сведения об авторах, реквизиты заявителя. В настоящее время чаще всего в качестве заявителя выступает фирма, сотрудники которой разработали ПО в рамках служебного задания. В этой связи необходимо заметить, что, во избежание конфликтных ситуаций, перед подачей заявки следует четко прояснить вопрос о принадлежности прав на программу и распределении их между работодателем (заказчиком) и разработчиком. Если же в качестве образца предоставлен исходный текст ядра программы, новая регистрация не требуется. После рассмотрения заявки и уплаты соответствующих сборов (в прошлом году размер их составлял от двухсот пятидесяти рублей до трех тысяч при срочном оформлении; в настоящее время сумма сборов не определена, хотя эксперты прогнозируют как минимум десятикратное повышение размера сборов) Агентство выдает заявителю свидетельство о регистрации с указанием ее даты, автора программы и правообладателя.

Степень надежности защиты

Как следует из законодательства и из практического опыта, охрана ПО с помощью авторского права не позволяет защитить его полностью. Отсюда следует необходимость комплексной защиты ПО с помощью различных институтов права, применяющихся по отношению к интеллектуальной собственности. Так, использующиеся в программе шрифты в некоторых случаях целесообразно защищать отдельно как "промышленный образец" (художественно-конструкторское решение изделия, определяющее его внешний вид). Другим примером может служить широкое при-

менение договорного права при отношениях по купле-продаже программных продуктов, позволяющего максимально учесть специфику той или иной сделки. Комплексное использование различных норм права значительно повышает защищенность программы и, следовательно, ее ценность.

Передача прав на программу

Такая передача осуществляется на основе лицензионного соглашения, в котором оговаривается по возможности широкий круг вопросов по ее использованию (в частности, количество архивных копий, количество одновременно работающих версий программы в сети, вознаграждение и тому подобное). При этом особую роль лицензионное соглашение играет в тех странах, где законодательство об охране программ отсутствует либо не развито (например, Беларусь, Украина). Условия такого соглашения зависят от специфики программного продукта и от особенностей его продаж (массовые или единичные). Кроме того, использование лицензионных соглашений позволяет торговать программами без уплаты налога на добавленную стоимость, в соответствии с пп. "К" п. 1 ст. 5 Закона о налоге на добавленную стоимость в редакции, установленной пп. "Г" п. 1 ст. 1 закона РФ "О внесении изменений и дополнений в отдельные законы РФ о налогах".

Особое место занимают лицензионные соглашения с дистрибьютерами и дилерами программ, где, помимо обычных условий использования, необходимо установить территориальные пределы распространения программы для отдельных дилеров, границы цен. Это поможет избежать неоправданной и неэффективной конкуренции между распространителями одного и того же программного продукта.

Защита и восстановление прав

К настоящему времени было мало прецедентов подачи судебных исков о защите авторских прав на программные продукты в России. Практика показывает, что возникающие в процессе создания и использования программ споры разрешаются на уровне межфирменных и внутрифирменных отношений (примером могут служить переговоры между известной западной фирмой и российским НИИ по поводу признания авторских прав на почти идентичный программный продукт под одинаковым названием. Эксперты прогнозируют возможность либо совместной деятельности, либо взаимных компенсаций). Это не в последнюю очередь объясняется высокой стоимостью судебных издержек, трудностями и медленным ходом судебного разбирательства. Однако не исключено возникновение в ближайшем будущем таких дел по отношению к особо злостным нарушителям авторских прав в показательно-воспитательных целях.

В заключение рассмотрим вопрос практической реализации своих прав на защиту программы. Возможны несколько вариантов.

Автор занимается защитой самостоятельно. При этом он экономит на стоимости консультационных услуг, но, учитывая необходимость высокой юридической квалификации для обеспечения должной защиты и отсутствие таковой у авторов в большинстве случаев, отметим, что самостоятельная защита может повлечь значительные убытки для автора.

Защиту предоставляет фирма, в которой работает автор. В этом случае особое значение играет грамотное составление договора между ним и работодателем о принадлежности имущественных прав на программу, что также требует юридических знаний.

Автор имеет возможность обратиться за консультациями в специализированные организации, занимающиеся вопросами правовой охраны программ, в частности, в Агентство, осуществляющее комплексное правовое обслуживание по желанию заказчика и имеющее региональные филиалы и отделения, оказывающие подобные услуги в различных регионах Российской федерации.

Юридическая поддержка программного продукта осуществляется специализированной фирмой. Такая фирма разрабатывает индивидуальный комплекс мер по регистрации и дальнейшему сопровождению программного продукта на всех этапах его использования. В настоящее время, несмотря на значительный потенциальный рынок таких услуг, на нем действует очень ограниченное количество таких специализированных фирм. В частности, интересен опыт Российской независимой службы промышленной и интеллектуальной собственности "ВЕСТА" — члена Ассоциации производителей программных продуктов. В сферу услуг, предоставляемых клиентам, входят консультации по использованию авторского права на программы, заключения о целесообразности совершения тех или иных сделок в отношении ПО, разработка мер по отношению к нарушителям авторского права на программу и решение спорных вопросов при использовании ПО. В сравнении с возможными убытками при неэффективной защите ПО и нарушениях авторских прав на него стоимость услуг специализированного юридического консультанта, определяющаяся на повременной основе, невысока и составляет от одной до полутора тысяч рублей за час работы консультанта. В среднем регистрация одной программы обойдется автору в двенадцать тысяч рублей.

Согласно прогнозам экспертов, значимость грамотной юридической защиты программ в России с течением времени будет возрастать, играя наряду с качеством самого программного продукта решающую роль в определении его рыночной ценности.

Автор выражает свою благодарность руководству Российской независимой службы промышленной и интеллектуальной собственности "ВЕСТА" за любезно предоставленные для статьи материалы.
Телефон службы: 267-84-22,
Печатников Анатолий Юрьевич. ■

ИМПОРТ, РАЗРАБОТКА И ПРОИЗВОДСТВО
ТЕЛЕКОММУНИКАЦИОННОГО
И КОМПЬЮТЕРНОГО ОБОРУДОВАНИЯ

НАТЕКС

Мы продаем :
- модемы и факс-модемы любых типов от 2400 до 14400bps производства Россия (НАТЕКС), США, Тайвань
- мультипорты 4RS232 (UNIX, XENIX, PC-MOS...), АЦП-ЦАП для IBM PC
- терминалы (VT-220, PC-Terminal)
- Multimedia карты (стерео звук + идеальное изображение)

У НАС ДЕЙСТВИТЕЛЬНО НИЗКИЕ ЦЕНЫ

ДЕЛАЙТЕ БИЗНЕС С НАМИ

Схема работы: (на примере самого „ходового" модема)

Название изделия	Характеристики изделия	Наша дилерская цена	Розничная цена
UNICOM 2400EV	внешний, 2400bps, V42bis & MNP5 h/w, рег. вых. уровня, сертификат МС РФ	$114 или рубли по курсу ММВБ	$155 (недорого, правда?)

ВАША ПРИБЫЛЬ: $41 с модема !

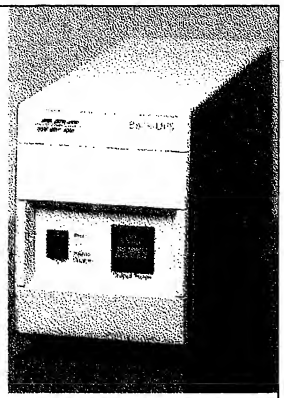
☎(095)324-40-00 FAX(095)324-28-07 ⊠103031 МОСКВА а/я 839

Журнал "Монитор" можно приобрести в городах России, Украины, Беларуси, Узбекистана, обратившись к нашим официальным дилерам.

1. **Самара**, *МП "Сети", директор Дмитрий Грецов*
   Тел. (8462) 354-932
2. **Волгоград**, *арендное предприятие "Техническая книга"*
   Тел. (8442) 363-597
3. **Чебоксары**, *фирма "Демос"*
   Тел. (8352) 208-395, 426-492, факс 209-437
4. **Киев**, *МП "Диас"*
   Тел./факс (044) 220-77-15, тел. 518-80-24
5. **Днепропетровск**, *Попов А.Е.*
   Тел. (0562) 428-593
6. **Минск**, *НПП "Триумф"*
   Тел. (0172) 661-609, 661-680, факс 666-606
7. **Ташкент**, *Центр молодежных исследований*
   Тел. (3712) 688-847
8. **Тамбов**, *АО "От Елены" ЛПС "Shift"*
   Тел. (0752) 22-44-63

Источники бесперебойного питания

СЛУЖАТ ДЛЯ ЗАЩИТЫ КОМПЬЮТЕРНОЙ СИСТЕМЫ ОТ ПЕРЕПАДОВ НАПРЯЖЕНИЯ И СБОЕВ В ЭЛЕКТРОСЕТИ



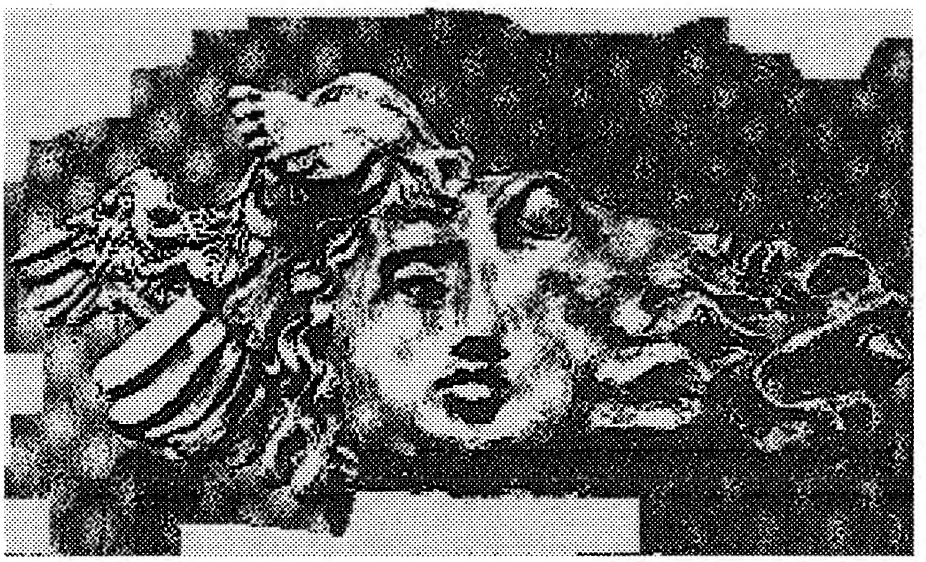
КАРАТ-2000

ОФИЦИАЛЬНЫЙ ДИСТРИБЬЮТЕР "AMERICAN POWER CONVERSION"

Телефон: (095) 200-1397, 200-1398
Телефакс: (095) 200-1393
Адрес: Москва, Садовая-Самотечная ул., 5

Региональные дилеры:

С.-Петербург	АО "СППКК"	515-2705
	ТОО "НТЦ МИТ"	298-0000
Барнаул	НПП "ВАРИАНТ"	26-04-51
Екатеринбург	АО "УРАЛКОМ"	44-09-93
Иркутск	ВП "ГРАДИЕНТ"	23-30-92
Калининград	НПК "СИМВОЛ"	21-52-74
Новосибирск	ТОО "МАЙКРОЛЭБ"	35-71-34
Сыктывкар	НПИКК "ГАРАНТ"	2-56-26
Тула	МНТП "МИРТЭКС"	31-78-72
Тюмень	АО "РЕГИОН"	25-05-25
Ульяновск	ВЭ ПКП "АГРОТЭК"	34-34-26
Уфа	НЦ "ВЕКТОР"	52-91-32



Игры с несколькими участниками

Часть 2

Продолжая серию материалов, посвященных исследованию игровых программ, мы печатаем вторую часть статьи, посвященной играм жанра ROLE PLAYING GAMES. Начало см. в журнале "МОНИТОР", 1993 год, № 2, стр. 74.

Исследование игрового пространства

Большой объем игрового пространства приводит к тому, что в отличие от адвентюрных игр каждая отдельная локация не предоставляет пользователю широких возможностей для исследования и приложения ума. Хотя определенная внимательность все же требуется. Так, например, могут быть обнаружены скрытые проходы. Более важной частью игры является обход доступных локаций и проникновение в те места, куда доступ затруднен. Сущность этих трудностей и способ их преодоления зависят от конкретной игры.

Так, во многих играх, где имеются игровые поля, занятые водой, горами, лесом и так далее, путешествие невозможно без соответствующей оснастки или без соответствующей подготовки персонажей посредством обучения. В игре "Might and Magic", например, небольшие проливы могут преодолеваться при помощи магии. В игре "Ultima V" фирмы ORIGIN персонажи могут добыть приспособление, позволяющее передвигаться по горам, что позволяет попасть в зоны, недоступные для прохода иным способом.

Но самым распространенным способом ограничения активности пользователя являются обычные запертые двери. Для проникновения в них нужны соответствующие ключи или ключевые предметы. Важные двери никогда не открываются с помощью отмычки. Ключевые предметы иногда нужны и для получения информации от встречных персонажей. Так, очень велик объем текстовой информации, которую можно узнать в процессе общения в играх серии "Ultima", хотя сам процесс общения там реализован так, что его трудно назвать диалогом.

Самые важные для прохождения игры предметы обычно достаточно труднодоступны, причем трудности связаны еще и с тем, что такие пред-

Attention !

ВНИМАНИЮ СИСТЕМНЫХ ОПЕРАТОРОВ:

При использовании печатных материалов в архивах BBS ссылка на источник публикации и указание авторства обязательны:
"ИНФОРКОМ",
121019 Москва, Г-19, а/я 16.

меты, как правило, располагают за заслоном противников, сквозь который приходится пробиваться с боем.

Ориентирование по сторонам света обычно предусмотрено во всех играх. Если в игре нет автоматического картографирования пройденного пространства, рекомендуется рисовать схемы лабиринтов вручную. Здесь работает правило: чем меньше общий объем игрового пространства, тем больше всевозможных хитростей. В играх "Eye of Beholder I и II" лабиринт состоит из многочисленных этажей, каждый из которых в плане представляет квадрат 32x32 локации. Составление плана позволяет обнаружить "подозрительные" места, где остаются неисследованные зоны. Очень часто в такие зоны возможен доступ через иллюзорные стены, сквозь которые можно пройти или же проход туда может быть открыт нажатием малозаметной кнопки на одной из стен.

Надо отметить, что в современных играх жанра RPG могут существовать различные альтернативные способы передвижения: пешком, с помощью технических средств, с использованием телепортирующих устройств или магии. Возможность разными способами попасть в то или иное место обеспечивает пользователю многовариантность стратегии, в зависимости от того, какими навыками обладают герои, включенные в состав экспедиции. Такая многовариантность в определенной мере может служить характеристикой развитости программы.

В играх этого жанра рекомендуется постараться посетить все доступные локации, даже если по внешним признакам там нет ничего интересного. Пользователя часто подстерегают неожиданности.

Поединки с противником

Употребляемое здесь слово "противник" выбрано с целью обозначить единым словом все, что создала мифология народов мира и дополнительно придумали авторы игровых программ. В каждой игре этого класса участвуют обычно не менее двух десятков типов различных чудовищ, монстров, странных созданий или просто враждебных персонажей, единственная цель которых — физически уничтожить отряд. По мере возрастания возможностей героев им становятся по силам все более опасные противники.

Можно рекомендовать следующие принципы, позволяющие героям избегать преждевременных встреч со слишком серьезным для них противником. Если игра разбита на последовательные уровни, то переходить к следующему рекомендуется только после полного исследования текущего. Исследование игрового пространства лучше начинать вокруг исходной точки и постепенно расширять сферу операций. Зоны, которые трудно отыскать или в которые трудно проникнуть, обычно грозят немалыми опасностями. И, наконец, если герои получают предупреждение об опасности, то такому предупреждению обычно стоит верить.

В большинстве игр при встрече с противником единственное, что остается героям, — это вступить в бой. Впрочем, иногда удается избежать этого боя и подкопить сил или подлечить персонажей, находящихся не в лучшем состоянии. Часто временное отступление бывает нужно из тактических соображений, а учитывая, что в этих играх редко наблюдается прессинг по времени, беспорядочное бегство все же предпочтительнее, чем гибель. Победа в бою приводит к накоплению боевого опыта, взятию военной добычи. Поражение — это гибель всех персонажей и необходимость начинать игру сначала или с отложенной записи. Во многих играх гибель нескольких членов отряда хоть и позволяет продолжать игру, но эквивалентна полному поражению (не забывайте сохранять текущее состояние, если не уверены в благоприятном исходе боя). Можно рекомендовать также входить в поединок с локаций, позволяющих своевременно воспользоваться возможностью отступления.

Обратите особое внимание на то, что многие игры этого жанра коварно записывают на диск состояние игры после Вашего поражения в бою. Поэтому, если Вы практикуете периодическое сохранение игры, целесообразно завести для файлов сохраненных состояний особый каталог на диске, недоступный игровой программе.

Вступая в сражение с противником, следует стараться представить себе, чего стоит ждать от успеха. Обычные результаты победы — это увеличение опыта героев, обеспечение прохода к новым локациям, а также взятие трофеев (некоторые предметы, важные для дальнейшего прохождения игры, могут быть получены только как трофеи в результате боя с хранителем или владельцем соответствующего предмета). Некоторые поединки знаменуют собой этапы прохождения игры (обычно это специальные поединки, например, штурм крепости в "King's Bounty" или особо выделенные противники, находящиеся в наиболее труднодоступных местах). В тех играх, где противники не появляются снова, уничтожение отряда врагов уменьшает общее количество противников и тем самым позволяет сделать более безопасными соответствующие локации.

Будьте внимательны: не во всех играх можно легко понять, друг или враг встретился Вам. Не всегда следует нападать на любую встречную группу. Кроме того, поскольку время от времени приходится встречаться с новыми, еще не знакомыми противниками, многие битвы таят в себе угрозу поражения. Поэтому, встретившись с противником, прежде всего следует решить, получите ли Вы пользу от поединка.

При реализации самого сражения создатели игр проявили большую фантазию, и формы этого сражения крайне разнообразны. В качестве же основных можно выделить, пожалуй, две основные формы: аркадную и тактическую.

В сражениях, реализованных в аркадной форме (встречаются реже), действие происходит в реальном времени и успех определяется в немалой степени бы-

стротой реакции, хотя подготовленность героев к бою, естественно, играет важную роль. В качестве примера можно указать игры "Hardnova" и "Eye of Beholder". Здесь герои взаимодействуют с противниками, находящимися в соседних локациях. Возможность удара, выстрела или нападения другим способом герои получают с некоторой частотой и задача пользователя нанести этот удар, как только предоставится возможность. Противники действуют таким же способом, но компьютер не упустит возможности нанесения удара. Степень поражения в результате атаки определяется текущими параметрами героев и противников. Немалую роль играет также и случайность. Это, кстати, приводит и к тому, что при повторении поединка он может сложиться и по-другому. Пользователь может нередко по ходу поединка приблизительно определить вероятность успеха и принять решение об отступлении или о повторном считывании ранее отложенной записи. Так он может попытаться повторить сражение с другим исходом либо вообще избежать этого поединка и вернуться к нему позже, в более подготовленном состоянии. Именно в этих играх бегство от противника с последующим восстановлением сил и возвращением может оказаться иногда лучшим способом постепенного перемалывания сил врага, в то время как прямая атака может привести лишь к неудаче.

В сражениях, реализованных в более распространенной тактической форме, герои и противники располагаются на некоторой "местности", особенности которой могут быть использованы в бою. Нередко при этом меняется и экранный интерфейс. Каждый персонаж получает ход по очереди, хотя возможно, что и с разной частотой. Ход представляет собой либо перемещение, либо нападение, либо комбинацию "перемещение-удар", либо использование магического заклинания (молитвы). Следует стараться понять, происходит ли действие в реальном времени или по тактам, поскольку динамическая графика может ввести в заблуждение. Так, внешняя форма поединка в "Eye of Beholder" и в "Might and Magic" практически одинакова, хотя в первой игре поединок аркадный, а во второй — тактический.

Простейшая форма такого тактического поединка — единоборство, встречающееся в тех играх, где действует один персонаж. Это в основном игры класса D&D. Здесь бой происходит в том же лабиринте, по которому путешествует герой. Основная тактическая задача — избежать опасности попадания в ловушку. Самая неприятная ситуация — когда герой попадает в окружение врагов и не может из него выбраться. На каждый нанесенный им удар следуют несколько ответных. Основной прием — сохранить благоприятные возможности маневра для выхода из боя, если он начнет принимать неблагоприятный оборот. Обычные способы — отступление, парализация противника, телепортация в безопасную зону.

Более интересны в тактическом плане игры, где происходит столкновение отряда с отрядом. Здесь, как правило, тактическое искусство позволяет выигрывать сражения, которые были бы заведомо проиграны, если бы противник обладал хотя бы небольшим тактическим мастерством или мог бы поставить перед собой задачу нанести Вашему отряду максимальный ущерб. Но компьютерный противник в поединках обычно туповат и прямолинеен, и Ваш отряд или герой для атаки выбирается случайно или по какому-либо слишком простому правилу. Для пользователя простейшим приемом является концентрированная атака, которая должна уменьшать число противников с тем, чтобы меньшее число противников могло атаковать героев во время очередного хода. Важными являются также возможности использования оружия дальнего действия и оружия массового поражения, позволяющего атаковать одновременно несколько противников. Как правило, это некоторое магическое оружие. Так, например, заклятие "Fireball" в игре "Pools of Darkness" захватывает одновременно 37 локаций зоны боя, поражая все, что на них находится. В игре "Martian Dreams" такого рода оружием является дробовик, стреляющий конусом дроби, также поражающим одновременно несколько локаций.

Противников, обладающих такого рода оружием, желательно вывести из строя сразу или бросить в поединок свои малоценные или, наоборот, высокостойкие, но маломаневренные войска. Следует также стремиться в первую очередь уничтожать противника, который представляет угрозу для вашего войска — близко расположенного или обладающего возможностью атаки издалека. В некоторых поединках какие-либо из противников имеют особые свойства. Так, например, в игре "Uncharted Waters" фирмы KOEI поединок представляет собой морское сражение, результат которого определяется состоянием флагманского корабля. Здесь следует любой ценой сконцентрировать свое нападение именно на нем.

Как правило, компьютерный противник не упускает возможности нанести удар по ближайшему из героев. Это позволяет в некоторых играх провести невыгодный для противников размен, связав его наиболее опасные силы бесполезным боем и обеспечив каким-то из своих войск возможность беспрепятственной атаки, особенно издалека.

Если в результате поединка Вы одержали победу, стоит посмотреть на изменение параметров героев и обратить внимание на сообщения, которыми поединок сопровождался. Анализ параметров иногда может показать, что то, что казалось блестящей победой, было на самом деле серьезной неудачей. Стоит внимательно проанализировать ситуацию и после поражения, чтобы понять, что было причиной неудачи: недостаток тактического мастерства, невезение или же недостаточная готовность и опытность героев. В некоторых случаях неудача может оказаться и необходимым шагом к успеху.

Игровая магия

Большинство игр этого класса предусматривает возможности применения магических заклинаний. Магическими способностями обладают два класса персонажей: маги и священники. Магические заклинания можно грубо разбить на три класса: боевая магия — заклинания, нацеленные на уничтожение противников в бою; информационная магия — заклинания, помогающие в исследовании игрового пространства; защитная магия — заклинания лечебные и дающие персонажам для боя магическую защиту. Не в каждой игре встречаются заклинания всех трех классов.

Заклинания, доступные магам и священникам, обычно существенно различаются. Хоть и те и другие могут иметь заклинания разных классов, маги более тяготеют к боевой магии, а священники — к защитной. Интересно отметить, что набор доступных заклинаний не так разнообразен, как можно было бы предположить. На самом деле практически одни и те же заклинания переходят из игры в игру, независимо от сюжета и фирмы-производителя. Такая историческая традиция безусловно помогает пользователю быстрее освоить новую игру, особенно если английский язык не является для него родным.

Для того чтобы употребить заклинание, персонаж должен иметь либо магический предмет (свиток, жезл, книгу, крест), либо знать это заклинание наизусть, а иногда требуется и то и другое. Действие заклинания может быть усилено при наличии некоторого предмета-концентратора, например, рубинового кольца, хрустального шара и тому подобного. Некоторые заклинания применимы только в бою, а некоторые — только в условиях путешествия. Все заклинания, изменяющие параметры и характеристики персонажей, действуют в течение ограниченного времени. В некоторых играх требуется время, чтобы выполнить заклинание, и, если персонажу помешают в бою, он может потерять и ход, и данное заклинание. Возможности магов и священников как по количеству, так и по качеству заклинаний возрастают с увеличением опыта.

Для того чтобы маг или священник выучил заклинание наизусть, обычно требуется время и соответствующие условия. Обычно это соответствует режиму "отдых", который бывает возможен не в любой момент и не в любом месте и также может прерываться вмешательством противников. Отдых обычно используется и для лечения персонажей (впрочем, при отдыхе здоровье постепенно восстанавливается автоматически).

Боевая магия играет особую роль в большинстве игр со сказочным сюжетом. Там, где встречающиеся отряды противников многочисленны, именно маги выигрывают бои, так как только они обладают оружием массового поражения, которое может действовать сразу на группу врагов. В этом случае в бою следует стремиться обеспечить возможность беспрепятственного использования своих магов, а также лишить активности или быстро уничтожить магов противника. Широкие возможности для применения тактического искусства открываются, например, в сериале "Forgotten Realms" фирмы WESTWOOD ASSOCIATES.

Стратегический аспект

Ограничение боевого отряда несколькими героями приводит к тому, что в этих программах тактический аспект явно доминирует над стратегическим. Для стратегических приемов остается узкое поле, и они сводятся к двум аспектам.

Во-первых, это стратегическое планирование. По результатам исследования игрового пространства пользователь должен наметить себе стратегический план с тем, чтобы проходить труднейшие участки с постепенным увеличением их сложности и с соответствующей подготовкой и оснащением героев.

Второй аспект заключается в том, что генеральную цель программ класса RPG можно рассматривать и как планомерное воспитание, обучение и оснащение своих героев, и если подходить к игре именно с таких позиций, можно совсем по-другому планировать и исследование игрового пространства, и проведение боевых сражений. Так, например, быстрое развитие характеристик персонажей может позволить пропускать без исследования игровые поля низших уровней сложности, что бывает важно в играх с ограниченным временем.

Обучающий аспект

Подход к RP Games как к играм, в которых генеральную цель представляет именно повышение квалификации персонажей, а не аркадный момент боевых сражений и не адвентюрный момент исследования игрового пространства, позволяет оценить возможность использования концепций этого жанра при разработке увлекательных обучающих программ.

В самом деле, заменив игровое пространство на пространство полей базы знаний, боевые поединки на тестирующие сеансы, а трофеи и сокровища на информационное наполнение образовательного характера, можно вполне реально подойти к созданию увлекательных обучающих программ. В них найдется место и для использования особенностей разных игровых амплуа героев, и для информационной магии, и для многого другого. Можно ожидать появления подобных программ в самое ближайшее время. ■

Школа ищет исполнителей для оснащения программами компьютерного класса на базе ПЭВМ ТУРБО-86 и МИКРО-86 по различным предметам для учащихся 9-11 классов.

С предложениями обращаться по адресу:
614051 г. Пермь, ул. Старцева, 9,
школа № 135.
Телефон: 48-03-60.

Нашел ли ты секрет игры "The Secret of Monkey Island"?

О. Стариков

Фирма-разработчик: *Lucas Arts Entertainment Company*
Графика: *CGA, MCGA, EGA, VGA*
Звук: *Internal, Tandy, Game Blaster, Adlib*
Управление: *Keyboard, Joystick, Mouse*

В первой части Ваш герой по имени Гайбраш Трипвуд (Guybrush Threepwood) пытается приобщиться к занятию, достойному настоящих мужчин, — стать пиратом. Но, как только он сдает вступительный экзамен на острове Мели (Melee island), привидение — капитан Ле Чак (Le Chuck) похищает правительницу острова. Гайбраш собирает команду для своего корабля и отправляется на выручку красавицы, зная, что пираты-привидения обитают на острове Обезьян (Monkey Island). Во второй части, приведя корабль к острову, герой мастерски использует пушку и попадает на берег заветного острова. Отсюда начинается третья часть игры. Здесь Гайбраш приручает обезьяну, знакомится с "робинзоном" по имени Герман Нутрут (Herman Noothroot), завоевывает доверие каннибалов-аборигенов и с помощью новых друзей попадает в катакомбы, где находится корабль-призрак. Взяв там магический корень, Ваш герой возвращается к аборигенам, которые изготавливают магический эликсир от привидений. Здесь начинается четвертая часть игры. Вновь спустившись к месту стоянки призрачного корабля, Гайбраш узнает, что коварный Ле Чак отправился обратно на остров Мели, чтобы жениться на своей пленнице. Герой опять пускается в погоню и настигает пирата в церкви, где уже началось венчание. Здесь у Вашего героя крадут эликсир, а капитан начинает вульгарно избивать всех, кто попадается ему на пути, но особое внимание уделяет, естественно, сопернику. Совершенно случайно Гайбраш попадает в автомат для грога, откуда вываливаются монеты и бутылка, в которой оказывается такой же магический эликсир. Использовав бутыль, Вы навсегда избавляетесь от коварного привидения и остаетесь с очаровательной правительницей.

Теперь я хочу предложить Вам некоторые подсказки для таких мест игры, над которыми я довольно долго ломал голову.

Тщательно переводите все разговоры вашего героя с персонажами игры, а также объявления, записи и так далее. В них Вы достаточно часто сможете обнаружить информацию, которая в той или иной степени необходима для успешного прохождения игры.

Итак, в первой части... Как только Вы зайдете на кухню в баре (Scumm Bar), Вы увидите за дверью рыбу. Но когда Вы выйдете, чтобы ее взять, сразу же прилетит чайка. Надо наступить на доску, на которой лежит рыба, чайка испугается и взлетит, но вскоре сядет обратно. С третьей или четвертой попытки Вы легко эту рыбу возьмете, а вернувшись на кухню, сварите ее в кипящей кастрюле (use fish in pot) и вытащите (pick up stewed fish). Этой рыбой Вы рассчитаетесь с троллем на мосту.

Деньги Вы возьмете в цирке (на карте его местоположение обозначено как clearing). Чтобы попасть на островок (на карте обозначен как island), используйте резинового цыпленка (use rubber chicken on cable).

Чтобы усыпить собак, Вам нужно мясо (возьмите его на кухне) и желтый лепесток (найдите в лесу желтые цветы). Положите лепесток на кусок мяса (use yellow petal with hunk of meat), и у Вас получится прекрасное блюдо (meat with of condiment) — собачий транквилизатор. Дайте полученный продукт собакам, и они уснут.

Заключенный поможет Вам добыть идола (fabuluos idol). Для того чтобы расположить узника, вначале дайте ему конфеты "Холодок" (breath mint), которые всегда есть в продаже в лавке хромого шкипера, и Отис (так зовут уголовника) разговорится.

После того как Вы побываете в доме правительницы, у Вас окажется сильнодействующее средство для отпугивания грызунов (gopher repellent). Дайте его заключенному и в обмен получите кусочек торта (cake). Надкусив (use cake), Вы извлечете оттуда напильник (file).

Чтобы выбраться из воды, подберите идола (pick up fabulous idol) и солидно, не торопясь, выходите на сушу.

Для покупки корабля Вам необходим кредит (credit), который можно получить у того же хромого шкипера. Для этого Вам необходимо отослать шкипера к учителю фехтования (sword master) и произвести некоторые манипуляции с ручкой сейфа (pull, push).

При покупке корабля надо поторговаться с продавцом.

При сборе команды не забывайте показывать документы (note).

Чтобы освободить заключенного, соберите все кружки в баре, налейте в одну из них грог из бочки и, переливая время от времени грог из плавящейся кружки в новую (use mug with melting mug), донесите и вылейте на замок камеры жуткий напиток (melting mug). После этого спешите в док.

Во второй части, чтобы сварить суп, Вам необходим рецепт, который находится в шкафу (cabinet). Возьмите коробку каши (cereal) и откройте ее (open cereal). В ней Вы найдете ценник (price), а вглядевшись в него (look at price), Вы получите ключ от шкафа (small key). Теперь рецепт у Вас, и Вы можете идти варить суп, для этого у Вас есть все составные части.

Чтобы стрелять из пушки, Вы, прежде всего, должны на камбузе сделать запал (use feather pen with fire). Затем из куска веревки Вам надо сделать фитиль (use large piece of rope with cannon) и заряд (use gunpowder with cannon nosle). Теперь Вам осталось использовать опыт барона Мюнхаузена (use pot), и Вы получаете прямой билет на остров Обезьян.

В третьей части, для того чтобы получить ядро и порох, просто толкните пушку (push cannon).

Чтобы взять весла, Вам нужны две веревки. Первую Вы найдете сразу, а вот со второй могут возникнуть сложности: найдите дамбу и, насыпав порох (use gunpowder with dumb), взрывайте дамбу (use noteworthy rock with cannonball), после чего Вы возьмете вторую веревку и сможете достать весла.

Для приручения обезьяны Вам нужны пять бананов. Первый Вы увидите сразу же возле банановой пальмы. Чтобы достать второй и третий, нужно использовать катапульту. Поднимитесь на самый верх горы и сбросьте камень (push rock), лежащий на самом краю. Теперь берите из кучи (heap of rocks) еще один камень, спускайтесь к катапульте и толкнув (push) или потянув на себя (pull) катапульту, попадайте в пальму. Четвертый и пятый бананы находятся у аборигенов-людоедов. Берите лодку и ищите их деревню.

Выбирайтесь из темницы через отверстие в полу. Открыть проход к огромной голове обезьяны (gigantic monkey head) Вам поможет прирученная обезьяна. На одном из тотемов (totem pole) потяните себя за нос (pull nose). Обезьяна повторит Ваши действия, и проход откроется.

Для того чтобы попасть через большую голову обезьяны в катакомбы, Вам нужен ключ, которым обладает "робинзон". Найдите рядом с гигантской головой (gigantic monkey head) фигурку маленького идола (small idol) и дайте его аборигенам, которые в знак благодарности оставят Вас в покое. Затем возьмите в своей темнице приспособление для сбивания бананов (banana picker) и обменяйте его на ключ от головы обезьяны (monkey head key).

Чтобы отыскать корабль-призрак, Вам потребуется проводник. Дайте людоедам листок (leaflet), и Вы получите взамен голову навигатора (head of navigator). Спускайтесь в катакомбы и с помощью головы (use head of navigator) ищите корабль.

Для того чтобы попасть на корабль-призрак (ghost ship), выполните некий магический ритуал с ожерельем (use necklace on navigator).

Чтобы найти магический корень (root), Вам потребуется выполнить такую последовательность действий:

- выдерните перо у цыпленка-призрака (pick up ghost chicken);
- пощекочите пером спящего пирата-призрака (use ghost feather with sleeping ghost crew) так, чтобы спящий выронил выпивку (jag of grog);
- в капитанской каюте с помощью магнитного компаса (use magnetic compass with key) добудьте ключ от трюма (hatch);
- чтобы достать смазку (grease), надо налить в миску грог (use jag of grog with dish). Крыса заснет, и смазка Ваша! Помажьте ею скрипящую дверь (use squeaky door on globe of grease) и возьмите инструменты (tools);
- после этого спускайтесь к люку в трюм и, потрудившись над решеткой (use tools with glowing crate), а затем просто заглянув туда (look at glowing crate), Вы получите корень (wodoo root);
- теперь возвращайтесь к Вашим друзьям-людоедам и отдайте им корешок (root). Они сделают Вам из корня волшебную газировку (magic seltzer bottle).

В четвертой части от Вас требуется совсем немного: как только Вы попадете в автомат для грога (grog machine) и едва Вас оттуда извлечет гнусный капитан Ле Чак, надо взять фирменное пиво (root beer — это разновидность того же самого эликсира) и вылить его на голову Ле Чака (use root beer on Le Chuck).

Ну, вроде бы все! Теперь Вы можете отправляться в прекрасный волшебный мир игры. Но имейте в виду, что эта статья далеко не исчерпывает все ситуации, с которыми Вам придется столкнуться во время игры. При затруднениях ставьте себя на место главного героя и проявляйте побольше смекалки и находчивости. Я желаю Вам отлично провести время в поиске секрета острова Обезьян, ведь игра полна самых неожиданных и смешных приключений. Счастливого пути!

Список команд

OPEN	— открыть;
WALK TO	— идти к;
USE	— применить (использовать);
CLOSE	— закрыть;
PICK UP	— взять;
LOOK AT	— посмотреть на;
PULL	— тянуть;
TALK TO	— говорить с;
TURN ON	— включить;
TURN OFF	— выключить;
PUSH	— толкать;
GIVE	— дать.

Математические сопроцессоры Intel — скорость и точность вычислений

Зачем нужны математические сопроцессоры? Ответ вроде бы очевиден — для ускорения математических расчетов. А вот кому они нужны? Ведь большинство пользователей ЭВМ не занимаются решением дифференциальных уравнений. Может быть, для наиболее популярных в нашей стране задач использование сопроцессора окажется не столь эффективным, как для сложных расчетов? Но табл. 1 показывает, что сопроцессор экономит Ваше время, а стало быть, и деньги при работе с базами данных, электронными таблицами, деловой и научной графикой.

Как видите, даже при расчете зарплаты сопроцессор внесет немалый вклад в повышение производительности Ваших программных средств.

Если же Вы используете программный комплекс моделирования трехмерной графики, например, AutoCAD, Ваша машина станет обрабатывать данные почти в пять (!) раз быстрее.

Для того чтобы установить сопроцессор, нужно сделать не так уж много: выбрать подходящую модель и установить ОДНУ микросхему в специальное гнездо на системной плате. Все программы, использующие математический сопроцессор, автоматически определяют его присутствие в системе.

Конкретный выбор модели сопроцессора определяется типом центрального процессора Вашей машины и ее тактовой частотой (табл. 2). Фирма Intel выпускает сопроцессоры, обеспечивающие взаимодействие со всеми типами процессоров серии 80x86.

Но почему предпочтение следует отдать продукции именно фирмы Intel? Прежде всего, потому что именно Intel разрабатывала все семейство микропроцессоров i80x86 и, следовательно, именно Intel лучше остальных фирм знает, как обеспечить максимальную эффективность математического сопроцессора в вычислительной системе.

Во-вторых, на сегодняшний день сопроцессоры фирмы Intel стали промышленным стандартом. Большинство компиляторов с языков высокого уровня (например, фирм Borland и Microsoft) поддерживают систему команд i80x87. Если же Вы решите использовать сопроцессор другой фирмы, то Вам может потребоваться специальное математическое обеспечение для его использования. Свыше 600 компаний-разработчиков программного обеспечения во всем мире ориентируются на математические сопроцессоры фирмы Intel, и уже более 2100 прикладных пакетов и программ активно используют возможности, предоставляемые i80x87.

Таблица 1

Сравнительное время выполнения тестовых программ

Программа	Краткое описание тестовой задачи	Время без сопроцессора	Время с сопроцессором Intel
LOTUS 1-2-3 Release 3	Программа расчета налогов	51	15
dBASE IV	Вычисление квадратного корня	78	27
Excel 3.0	Вычисление тригонометрических функций	67	41
Statgraphics 5.0	Расчет регрессии	36	20

Таблица 2

Какой сопроцессор нужен моему компьютеру?

i486DX	Встроен в центральный процессор
i486SX	i487SX
i386DX (16, 20, 25, 33 МГц)	i387DX или RapidCAD
i386SX Portable (16, 20, 25 МГц)	i387SL Mobile
i386SL Portable (16, 20, 25 МГц)	i387SL Mobile
i386SX 25 МГц	i387SX-25
i386SX 20 МГц	i387SX-20
i386SX 16 МГц	i387SX-16
Все модели i80286	i287XL
i8088 или i8086 (10 МГц)	i8087-1
i8088 или i8086 (8 МГц)	i8087-2

В-третьих, Intel проявляет постоянную заботу о пользователях, разрабатывая новые модели сопроцессоров, повышающие производительность уже существующих систем. Примером такой разработки может служить инженерный процессор RapidCAD, предназначенный для использования в системах на базе i80386DX.

Intel RapidCAD представляет собой набор из двух микросхем, которые устанавливаются ВМЕСТО процессора i80386DX и сопроцессора i80387DX. Первый кристалл совмещает как процессор, так и сопроцессор. При этом за счет реализации двух устройств на одном кристалле исключаются задержки, связанные со временем распространения сигнала между кристаллами i80386DX и i80387DX. Операции над целыми числами в Rapid CAD также выполняются быстрее. Второй кристалл в наборе RapidCAD — это просто заглушка, устанавливаемая в гнездо математического сопроцессора и предназначенная для обеспечения совместимости с традиционными системами на базе i80386 по обработке немаскируемых исключений (exceptions) при операциях с плавающей запятой.

Общее повышение производительности по сравнению с i80387DX (Rapid CAD уверенно работает с частотами от 16 до 33 МГц) составляет в среднем 65 процентов.

И, наконец, последнее, но очень важное преимущество микроэлектронных изделий производства фирмы Intel. Это гарантия на весь срок службы микросхем и бесплатная поддержка пользователей.

Приобрести микроэлектронные изделия фирмы Intel сегодня можно у дилеров фирмы, узнав их адреса и телефоны в московском представительстве фирмы Intel по телефону: (095) 445-40-01.

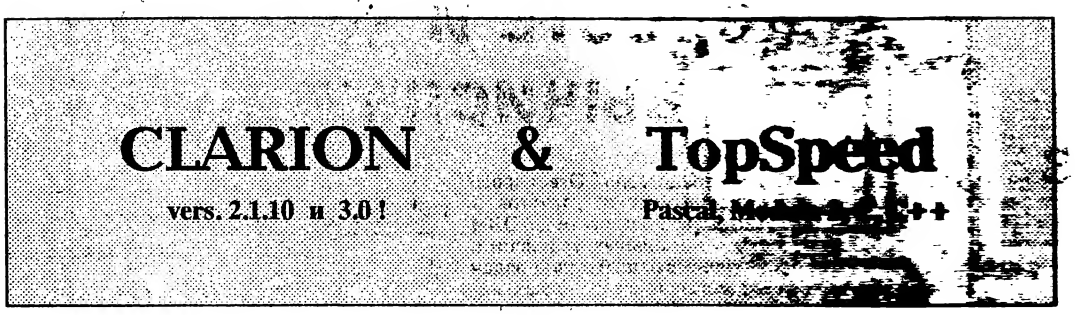
Торговые марки:

RapidCAD, i386, i387, i287, i486, i487 являются торговыми марками корпорации Intel.

Остальные названия упомянутых в статье продуктов являются торговыми марками соответствующих фирм-производителей. ■

ИНСАЙТ INSIGHT

Дистрибьютер CLARION SOFTWARE Corp.



Идеальные средства разработки заказного программного обеспечения.

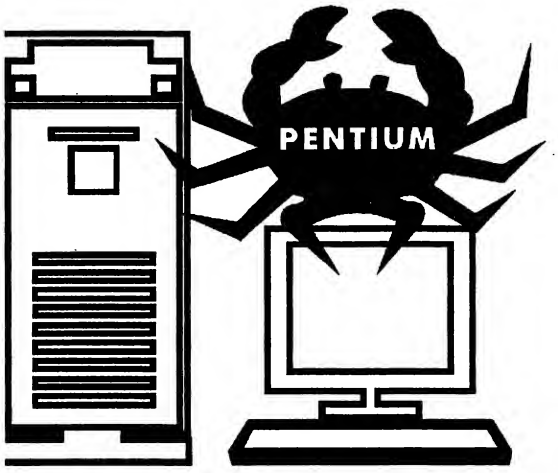
Учебные курсы, семинары CLARION User Group,
для официальных пользователей - "горячая линия".

CLARION DataBase Developer ver. 3.0:

- Высокоэффективный компилятор, выполненный по TopSpeed-технологии, создающий стандартные OBJ - файлы, интерактивный отладчик VID;
- SmartLink, DLL библиотеки, динамические оверлеи, DOS Extender;
- Драйверы баз данных (Btrieve, NW SQL, Paradox, dBase, Fox, Clipper);
- Словари баз данных;
- Template Language - язык описания процедур Designer'a;
- Поддержка мыши, специальные экранные эффекты, SAA/CUA-интерфейс, вертикальный и горизонтальный скроллинг, отображение PCX и GIF файлов
- Новая версия Report Writer.

Предлагаем также программное обеспечение фирм
Borland, Symantec, Microsoft, Corel, WordPerfect
Модули расширения памяти **SIMM, SIP**
Принтеры **EPSON**

103489, Москва, Зеленоград, корп. 602а
тел. 536-69-00 факс 534-93-46
E-Mail: puh@insight.pccentre.msk.su | 2:5020/11
CompuServe: 100140,1224



Микропроцессор Pentium

Д. Бройтман

Часть 1. Архитектура

Микропроцессор Pentium относится к пятому поколению процессоров фирмы Intel. Он полностью совместим на уровне двоичного кода с процессорами Intel предшествующих четырех поколений: 8086/8088, 80286, Intel 386DX, Intel 386SX, Intel 486DX, Intel 486SX и Intel 486DX2. 22 марта 1993 года начались промышленные поставки двух моделей Pentium с тактовой частотой 60 МГц и 66 МГц.

В первой части настоящей статьи рассматриваются особенности архитектуры Pentium, отличающие его от процессоров Intel 486. Вторая часть будет посвящена новым возможностям Pentium, относящимся к программированию.

Полное и точное описание процессора Pentium содержится в фирменной документации [1].

1. Архитектура — краткий обзор

Перечислим основные архитектурные нововведения, появившиеся в Pentium.

1) "Суперскалярность". Это означает, что за один такт процессора может быть выполнено более одной команды.

2) Предсказание переходов. Специальные логические схемы пытаются предсказать, в какую точку программы будет передано управление, и заранее готовят к выполнению соответствующие фрагменты кода.

3) Конвейер для выполнения операций с плавающей точкой.

4) Новые алгоритмы обработки некоторых команд сокращают количество тактов процессора, необходимых для их выполнения.

5) Две разновидности кэш-памяти — кэш для кода и кэш для данных, каждая размером 8 Кбайтов.

6) Поддержка протокола обратной записи MESI при работе с кэш-памятью данных.

7) Внешняя шина данных размерностью 64 бита.

8) Конвейерная обработка циклов шины данных.

9) Дополнительный контроль за целостностью данных: проверка на четность адресов и внутренних массивов данных. Проверка на четность обеспечивает обнаружение ошибок в 53% компонент на кристалле Pentium, не уменьшая скорость выполнения команд.

10) Контроль с помощью функциональной избыточности. Это означает дублирование вычислений вторым процессором-"контролером", который выдает сигнал ошибки при расхождении значений на выходах процессоров. Функциональная избыточность обеспечивает практически 100% гарантию обнаружения ошибок.

11) Трассировка выполнения команд. Специальные контакты и циклы шины дают возможность с помощью внешней аппаратуры отслеживать потоки команд внутри процессора.

12) Дополнительные возможности тестирования. Поддерживается стандарт IEEE 1149.1 ("Test Access Port and Boundary Scan").

13) Расширенная реализация архитектуры режима управления системы (System Management Mode).

14) Изменения в реализации виртуального режима 8086, увеличивающие производительность работы в этом режиме.

Структурная схема процессора Pentium на архитектурном уровне приведена на рис. 1.

На схеме два конвейера команд. U-конвейер может выполнять все целые команды и команды с плавающей точкой; V-конвейер — простые целые команды и FXCH — команды с плавающей точкой.

Кэш-память команд, буфер переходов (BTB) и буферы предвыборки обеспечивают подготовку команд для выполнения. Команды выбираются из кэш-памяти команд или с внешней шины. Адреса переходов запоминаются в буфере переходов. Устройство де-

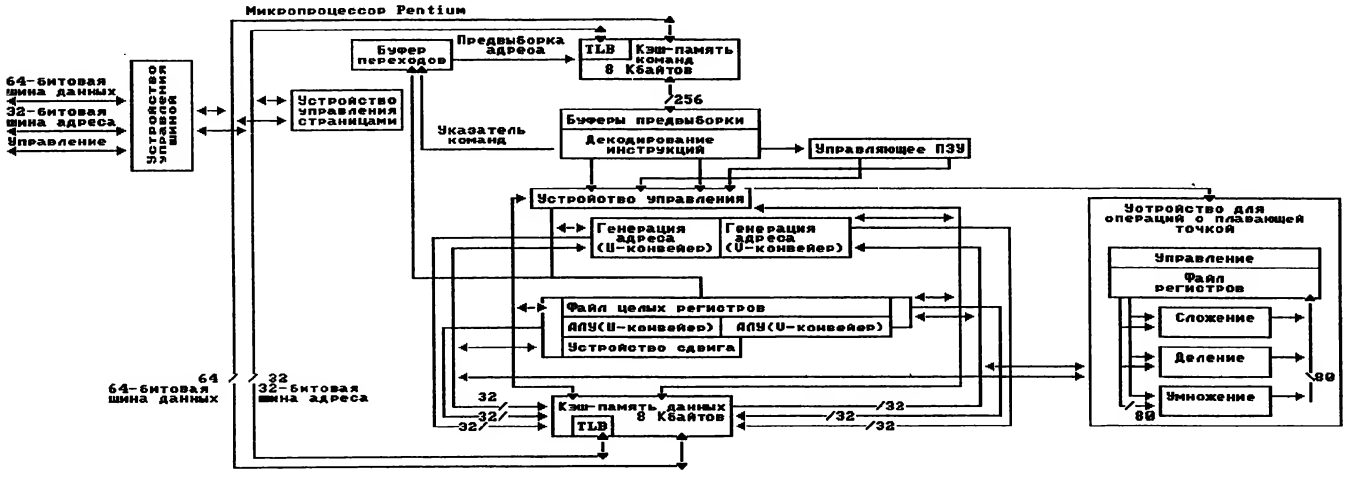


Рис. 1. Структурная схема процессора Pentium

кодирования преобразует выбранные команды так, чтобы они могли быть исполнены процессором. Управляющая память (Control ROM) содержит микрокод, управляющий последовательностью операций, осуществляемых в процессоре при выполнении команд. Она непосредственно управляет обоими конвейерами.

2. Конвейер и поток команд

Стадии конвейера

Как и в Intel 486, целочисленные команды проходят 5 стадий конвейера:

1) PF — предвыборка команд (Prefetch);
2) D1 — декодирование команды (Instruction Decode);
3) D2 — генерация адреса (Address Generate);
4) EX — выполнение (Execute);
5) WB — запись результата (Write Back).

На рис. 2 изображено прохождение команд через конвейер процессора Intel 486.

В Pentium два 5-стадийных конвейера могут работать параллельно и выполнять две целочисленные команды за машинный такт. На рисунке 3 изображен поток команд в процессоре Pentium.

Конвейеры в процессоре Pentium называются U- и V-конвейерами, а процесс параллельного выполнения двух команд — спариванием (pairing). U-конвейер может выполнять любую команду; V-конвейер — только так называемые простые команды, определяемые специальными правилами спаривания команд. При спаривании V-конвейер всегда выполняет команду кода, следующую за командой, выполняемой U-конвейером.

Основная структура конвейера та же, что и в Intel 486, однако она оптимизирована для достижения большей пропускной способности.

На первой стадии (PF) осуществляется предвыборка команд из кэш-памяти или оперативной памяти. Поскольку в Pentium реализованы отдельные кэш-памяти для команд и для данных, ликвидирован конфликт между обращениями к кэш-памяти за командами и за данными. Если требуемой команды нет в кэш-памяти, происходит обращение к оперативной памяти.

В предвыборке команд участвуют два независимых 32-байтовых буфера предвыборки и буфер переходов. В каждый момент времени предвыборку осуществляет только один из буферов предвыборки. Предвыборка выполняется последовательно до тех пор, пока не будет выбрана команда перехода. В этот момент буфер перехода предсказывает, будет или не будет осуществляться переход. Если предсказано, что перехода не будет, то продолжается последовательная предвыборка. Если предсказан переход, то включается другой буфер предвыборки, который начинает выбирать команды, начиная с той, на которую должен произойти переход. Если переход предсказан ошибочно, то конвейеры команд очищаются и предвыборка начинается снова.

На второй стадии (D1) два параллельных дешифратора пытаются декодировать и отправить на выполнение следующие две последовательные команды.

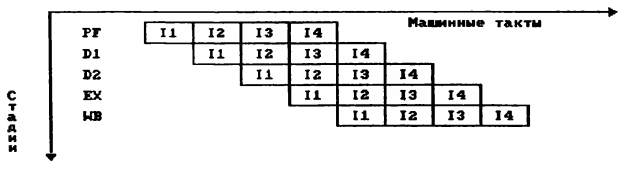


Рис. 2. Прохождение команд через конвейер Intel 486

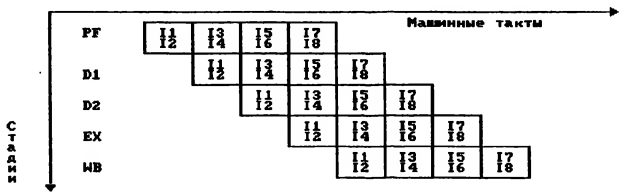


Рис. 3. Прохождение команд через конвейер Pentium

Дешифраторы определяют, одна или две команды будут выполняться, в соответствии с правилами спаривания команд.

На третьей стадии (D2), так же как и в Intel 486, определяются адреса операндов в памяти. В Intel 486 команды, содержащие смещение и непосредственный операнд, а также команды, включающие базовый и индексный режимы адресации, требовали дополнительного такта на декодирование. В Pentium эти ограничения отсутствуют, и такие команды выполняются за один такт.

На четвертой стадии (EX) Pentium, так же как Intel 486, осуществляет операции в АЛУ и обращение к кэш-памяти данных. Команды, требующие и того и другого, задержатся на этой стадии более чем на один такт. На этой стадии все команды в U-конвейере и все команды, за исключением условных переходов, в V-конвейере проверяются на правильность предсказания перехода. Микрокод в Pentium спроектирован так, чтобы использовать оба конвейера, поэтому команды, вызывающие микрокод, выполняются быстрее, чем в Intel 486.

На последней стадии (WB) команды могут изменить состояние процессора и завершают выполнение. На этой стадии на правильность предсказания перехода проверяются условные переходы в V-конвейере.

Правила спаривания команд

Для того чтобы две команды могли выполняться параллельно, они должны удовлетворять следующим условиям:

- обе команды должны быть "простыми" (что это означает, будет сказано далее);
- между ними не должно быть регистровых зависимостей типа "запись после чтения" или "чтение после записи";
- ни одна из команд не может содержать одновременно смещение и непосредственный операнд;
- команды с префиксами (кроме 0F в командах JCC) могут появляться только в U-конвейере.

"Простыми" являются следующие целочисленные команды:

1) mov	reg, reg/mem/imm	Команды
2) mov	mem, reg/imm	пересылки
3) alu	reg, reg/mem/ imm	Арифметические
4) alu	mem, reg/imm	и логические операции
5) inc	reg/mem	Увеличение и
6) dec	reg/mem	уменьшение на 1
7) push	reg/mem	Операции
8) pop	reg	над стеком
9) lea	reg, mem	Загрузка
10) jmp/call/jcc near		Команды передачи управления
11) nop		Пустая команда

"Простые" команды выполняются аппаратно, не требуют выполнения микрокода и исполняются, как правило, за один такт. Исключение: команды ALU mem, reg и ALU reg, mem, которые требуют для выполнения два и три такта соответственно, и их выполнение как "простых" обеспечивается специальными компонентами процессора.

Условные и безусловные переходы могут выступать только как вторая команда в паре. SHIFT на 1, ROT на 1, SHIFT imm могут быть лишь первой командой в паре.

Регистровые зависимости, которые запрещают спаривание, включают неявные зависимости через регистры и флаги. Например, ALU команда в U-конвейере (устанавливает флаги) не может быть в паре с использующими флаги командами в V-конвейере (например, ADC — сложение с переносом или SBB — вычитание с займом). Два исключения из этого правила — это последовательность "сравнение-переход" и пары из команд push и pop. Во втором случае для исключения неявной зависимости от указателя стека добавлены специальные аппаратные компоненты, чтобы эти частые операции могли выполняться параллельно.

Как правило, две "спаренные" команды могут выполняться параллельно и независимо. Исключение: пара команд типа "прочитать-изменить-записать", то есть арифметико-логические команды с операндом в памяти. Если две такие команды спарены, то к необходимым для их выполнения 3 тактам добавится задержка в два такта.

Для программиста выполняемые параллельно команды ведут себя точно так же, как если бы они выполнялись последовательно.

Комбинация из спаривания команд и предсказания переходов существенно повышает производительность процессора. В качестве примера рассмотрим базовый цикл из классической тестовой программы поиска простых чисел ("Решето Эратосфена").

```
for (k = i + prime; k <= SIZE; k+=prime)
   flags [k] = FALSE;
```

После компиляции будет сгенерирован следующий код на ассемблере (prime размещено в ecx, k — в edx, al содержит FALSE):

```
loop:
   mov  byte ptr flags [edx], al
   add  edx, ecx
   cmp  edx, SIZE
   jle  loop
```

Каждая итерация этого цикла займет 6 тактов процессора Intel 486. В Pentium будут спарены команды mov с add, cmp с jle. С предсказанием переходов каждая итерация цикла будет выполнена за 2 такта.

3. Внутренняя кэш-память

Внутри микропроцессора Pentium размещены кэш-память команд и кэш-память данных. Кэш-память

увеличивает производительность системы, поскольку чтение из кэш-памяти внутри процессора осуществляется быстрее, чем обращение к основной памяти через внешнюю шину. Уменьшается также загрузка внешней шины при многократном обращении по одному и тому же адресу.

Наличие раздельных кэш для команд и для данных дает возможность одновременного доступа к ним: за один такт могут быть произведены два обращения к данным и прочитаны 32 байта кода.

Кэш данных и кэш команд содержат по 8 Кбайтов и разделены на строки размером 32 байта (256 битов). Внешняя кэш-память не должна использовать строки меньшего размера.

Каждой строке кэш соответствует выровненный блок основной памяти размером 32 байта (последние 5 битов в адресе такого блока — нули). При обращении к фрагменту основной памяти меньших размеров происходит кэширование всего содержащего его 32-байтового блока.

Работа через кэш может происходить с любым фрагментом основной памяти, однако существуют и программная и аппаратная возможности запрета на кэширование определенных участков памяти. Если и программное обеспечение, и аппаратура допускают кэширование участка памяти, к которому произошло обращение, то процессор считывает 32-байтовую строку в соответствующую кэш-память. Перенос в кэш-память отсутствующего в ней фрагмента происходит только при чтении. При записи в участок памяти, отсутствующий в кэш, данные помещаются в специальный буфер, откуда посылаются в память через внешнюю шину, когда эта шина освобождается.

Кэширование используется во всех режимах процессора: реальном, защищенном и режиме эмуляции 8086. В правильно спроектированной однопроцессорной системе после включения кэширования при инициализации системы оно не требует специального контроля.

Кэш-согласование. Протокол MESI

Использование кэш-памяти в мультипроцессорных системах требует принятия специальных мер предосторожности. Дело в том, что когда один из процессоров обращается к фрагменту памяти, с которым другой процессор работает через свою кэш-память, первый может получить неправильные данные. Процессор Pentium обеспечивает специальные механизмы, которые гарантируют соблюдение так называемого кэш-согласования (cash consistency). Кэш-согласование означает, что если один из процессоров, входящий в систему из нескольких процессоров, и, возможно, нескольких внешних устройств кэш-памяти, изменяет какие-либо данные, то все остальные процессоры при обращении к этим данным получат измененные данные.

Кэш-согласование достигается с помощью использования протокола MESI (modified/exclusive/shared/invalid). Согласно этому протоколу, каждой строке в кэш-памяти данных присваивается одно из четырех состояний (физически это означает, что на каждую 32-байтовую строку в кэш-памяти данных приходятся два бита статуса). Состояние строки может быть изменено как процессором, содержащим эту кэш-память, так и другими устройствами, подключенными к внешней шине. По состоянию строки можно определить, действительна ли она (то есть соответствует ли ей строка в оперативной памяти), доступна ли она в других устройствах кэш-памяти и была ли она изменена.

Определение состояний

M — Modified (измененная). Строка доступна только в одном устройстве кэш-памяти и была изменена (то есть ее содержимое отлично от содержимого соответствующего ей фрагмента основной памяти). Доступ к M-строке (и чтение, и запись) может быть осуществлен без обращения к основной памяти через внешнюю шину.

E — Exclusive (исключительная). Строка также доступна только в одном устройстве кэш-памяти, но она не изменялась (то есть ее содержимое совпадает с содержимым соответствующего ей фрагмента основной памяти). Доступ к E-строке (чтение или запись) также может быть осуществлен без обращения к внешней шине. Запись в E-строку переводит ее в состояние M.

S — Shared (разделенная). Возможно, что эта строка присутствует более чем в одном устройстве кэш-памяти. Чтение S-строки не приводит к действиям на шине, но запись в S-строку генерирует цикл записи через внешнюю шину в основную память. При этом та же строка в других устройствах кэш-памяти может стать недействительной.

I — Invalid (недействительная). Строка недоступна в кэш-памяти. Чтение I-строки приводит к считыванию соответствующего фрагмента в кэш из основной памяти. Запись в I-строку генерирует цикл записи через внешнюю шину в основную память.

Протокол MESI определяет алгоритм, согласно которому строки кэш переходят из состояния в состояние.

Кэш-согласование для кэш-памяти команд поддерживается с помощью подмножества MESI протокола. Строки в кэш-памяти команд могут находиться только в двух состояниях: Shared и Invalid. Это связано с тем, что запись в кэш-память команд запрещена во избежание некорректного изменения кода. При попытке записи в кэш-память команд соответствующая строка в ней переводится в состояние I (недействительная).

Алгоритмы обновления данных

При записи данных в кэш Pentium может придерживаться одного из двух алгоритмов обновления дан-

ных (к чтению данных эти алгоритмы отношения не имеют).

1) Сквозная запись (write-through). Запись в строку кэш-памяти приводит к обновлению и кэш, и основной памяти. Сквозная запись используется, например, в графических приложениях, когда память должна обновляться сразу, чтобы изменения отразились на дисплее.

2) Обратная запись (write-back). При записи обновляется только строка в кэш-памяти. Алгоритм обратной записи сокращает трафик на внешней шине за счет исключения ненужных записей в основную память. Измененная строка кэш переписывается в основную память с помощью операции обратной записи, которая производится либо если строка удаляется из кэш (например, кэш-память полна и строка должна уступить место другой), либо если это необходимо для поддержки кэш-согласования.

Каждый участок в основной памяти может взаимодействовать с кэш по любому из алгоритмов обновления. Задать алгоритм для конкретного участка памяти можно и программным путем, и аппаратно.

Алгоритм замещения строк в кэш-памяти

И кэш данных, и кэш команд придерживаются так называемого LRU алгоритма (least-recently-used) замещения строк в кэш-памяти. Если в кэш есть недействительные строки, то новая строка будет помещена вместо одной из них. Если недействительных строк нет (кэш-память полна), то новая строка замещает строку, обращение к которой происходило наиболее давно.

Возможности управления работой кэш-памяти

Функционирование как внутренней, так и внешней кэш-памяти невидимо для прикладных программ. Однако знание алгоритмов работы кэш может быть использовано для повышения производительности программного обеспечения. Например, знание размеров кэш-памяти и алгоритмов замещения позволяет оптимизировать обработку больших массивов данных определенной структуры. В мультипроцессорных системах поддержка кэш-согласования также требует в некоторых случаях вмешательства системного программного обеспечения. Для таких ситуаций в Pentium предусмотрены специальные привилегированные команды управления кэш-памятью. Программы могут также воздействовать на поведение кэш, изменяя определенные биты в управляющих регистрах, в каталоге страниц и в таблицах страниц второго уровня.

Внешние устройства могут воздействовать на поведение кэш-памяти, подавая сигналы на специально предусмотренные для этой цели контакты.

4. Устройство для операций с плавающей точкой

Устройство для операций с плавающей точкой полностью перепроектировано по сравнению с Intel 486. Теперь оно представляет собой 8-стадийный конвейер и может выполнять одну команду с плавающей точкой за такт.

Как правило, команды с плавающей точкой не могут выполняться параллельно с целочисленными командами. Однако существует одно важное исключение. Многие команды с плавающей точкой могут быть выполнены только над операндом, находящимся в вершине стека. Для обмена данными с вершиной стека предназначена команда FXCH. Например, квадратный корень из числа, содержащегося в третьем сверху регистре от вершины стека, может быть вычислен с помощью последовательности команд

```
FXCH ST (3)
FSQRT
FXCH ST (3)
```

Так вот, эта часто используемая при вычислениях с плавающей точкой команда FXCH может выступать в качестве второй команды в паре, если ей предшествует одна из команд: FLD single/double, FLD ST [i], FADD, FSUB, FMUL, FDIV, FCOM, FUCOM, FTST, FABS, FCHS. "Спаренная" таким образом команда FXCH не требует времени на выполнение (0 тактов).

Новые алгоритмы обработки операций с плавающей точкой увеличивают скорость выполнения обычных операций (ADD, MUL, LOAD), по меньшей мере, втрое.

5. Внешняя шина данных

Размерность внешней шины данных в Pentium увеличена вдвое, по сравнению с Intel 486 — до 64 битов. Кроме того, введено конвейерное выполнение циклов передачи данных, так что два цикла шины могут одновременно находиться на выполнении. Скорость передачи данных по шине данных Pentium при тактовой частоте 66 МГц составляет

8 байтов x 66 МГц = 528 Мбайтов/сек,

что более чем в 3 раза превосходит максимальную скорость передачи в 50 МГц Intel 486 (160 Мбайтов/сек).

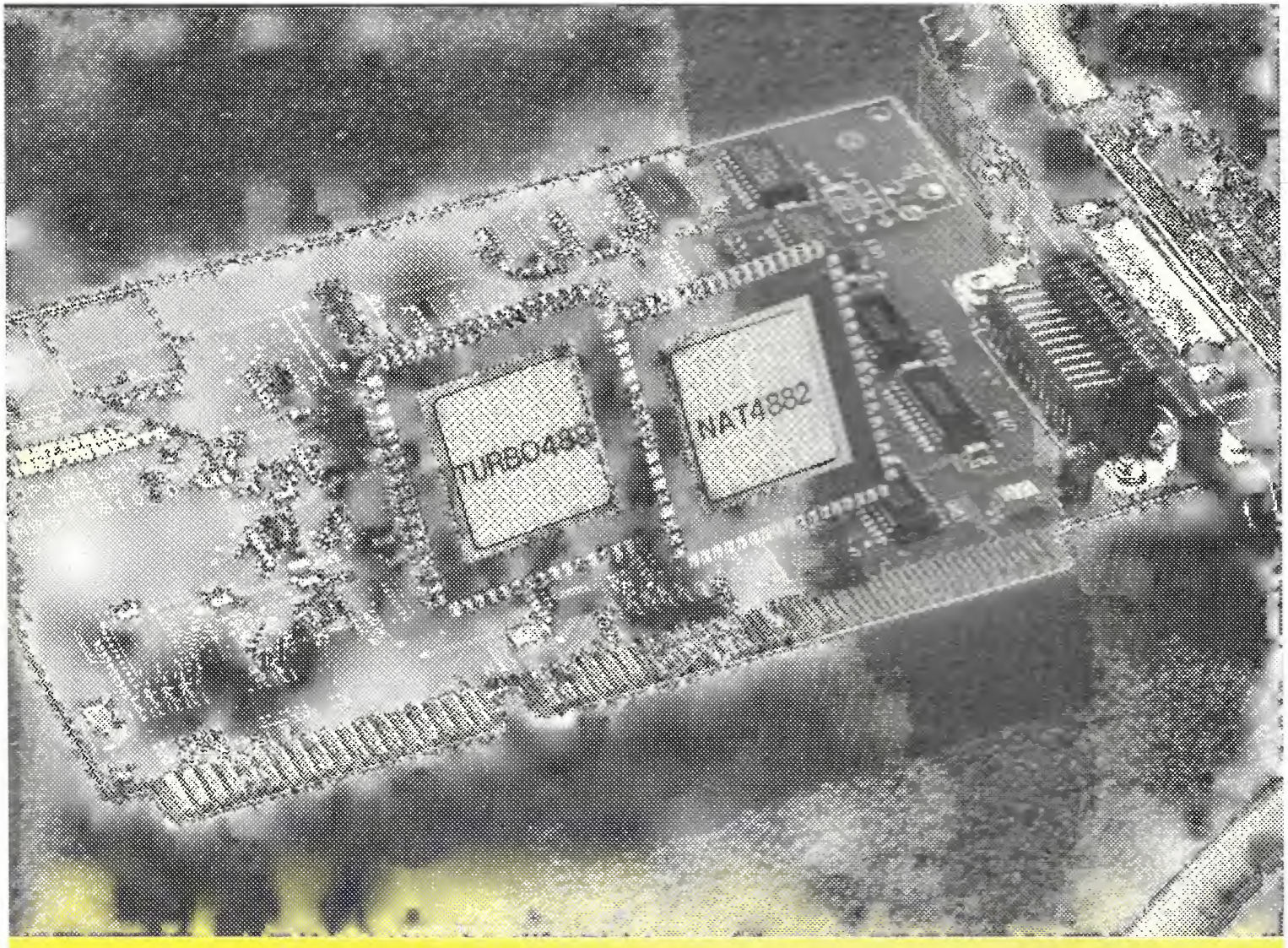
6. Общая оценка производительности

Интегральным результатом перечисленных выше архитектурных нововведений стал рост производительности в среднем в 3-5 раз (в 5-10 раз для приложений, интенсивно использующих операции с плавающей точкой), по сравнению с 33 МГц Intel 486DX и в 2,5 раза, по сравнению с 66 МГц Intel 486DX2.

Автор выражает благодарность сотрудникам московского представительства Intel Corporation за содействие при подготовке статьи. ■

Литература

1. Pentium Processor User's Manual: vol. 1-3. — Intel Corporation, 1993.



ФИРМА NATIONAL INSTRUMENTS (США) — МИРОВОЙ ЛИДЕР В ОБЛАСТИ ЗАВЕРШЕННЫХ ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРНЫХ СИСТЕМ ПО СБОРУ И ОБРАБОТКЕ ДАННЫХ

ПРЕДСТАВЛЯЕТ ВЫСОКОСКОРОСТНОЕ РЕШЕНИЕ В РАМКАХ СТАНДАРТА IEEE 488.2

IEE 488.2 является набором SCPI (Standard Commands for Programmable Instruments) инструментов и нового поколения GPIB–тестовых систем. Наш интерфйс AT-GPIB и программное обеспечение NI-488.2™ являются незаменимыми средствами использования IEEE 488.2 в Вашей работе.

Наше программное обеспечение NI-488® является признанным *de facto* стандартом отрасли. Последнее программное обеспечение фирмы в этой области NI-488.2 совместимо с существующими прикладными пакетами. AT-GPIB поставляется с версией NI-488.2 для DOS и Windows. Поставки могут также осуществляться и для OS/2 и UNIX.

По всем вопросам обращаться:
в МЕЖДУНАРОДНЫЙ КОМПЬЮТЕРНЫЙ КЛУБ (МКК). Тел. в Москве (095) 921–0902, (095) 925-4667. Факс (095) 921–0902,
а также к **дистрибьютерам** фирмы NATIONAL INSTRUMENTS в России:
ИНТЕЛЛЕКТУАЛЬНЫЕ СИСТЕМЫ (является также представителем). 129301, Москва, Проспект Мира, 186, строение 2. Телефон (095) 286–2438.
ТРАНСЭЛЕКТРО. 196070, Ст. Петербург, Почтовый ящик 330. Тел. (812) 294–0635, (812) 294–0443.

NATIONAL INSTRUMENTS®

Ваша программа, на Вашем компьютере, но гораздо быстрее!

Работу с этими и другими программами ускоряют сопроцессоры Intel

Математические сопроцессоры фирмы Intel:

— **привлекают крупнейших разработчиков программного обеспечения.** Свыше 600 таких компаний во всем мире ориентируются на математические сопроцессоры фирмы Intel, и уже свыше 2100 прикладных программ активно их используют;

— **предлагают проверенное решение для любого компьютера от XT до i486 SX.** Математические сопроцессоры фирмы Intel являются индустриальным стандартом. Во всем мире буквально миллионы пользователей с удовлетворением пожинают плоды повышенной производительности;

— **обеспечивают совместимость.** Поскольку сопроцессоры фирмы Intel разработаны специально для работы совместно с микропроцессорми Intel, на основе которых созданы все персональные компьютеры, они обеспечивают 100%–ную совместимость;

— **означают качество фирмы Intel.** Математические сопроцессоры, изготовленные фирмой Intel, являются наивысшим стандартом в мире и имеют пятилетнюю гарантию;

— **легко устанавливаются.** Надо лишь вставить сопроцессор в отведенное для него гнездо – и готово!

intel®